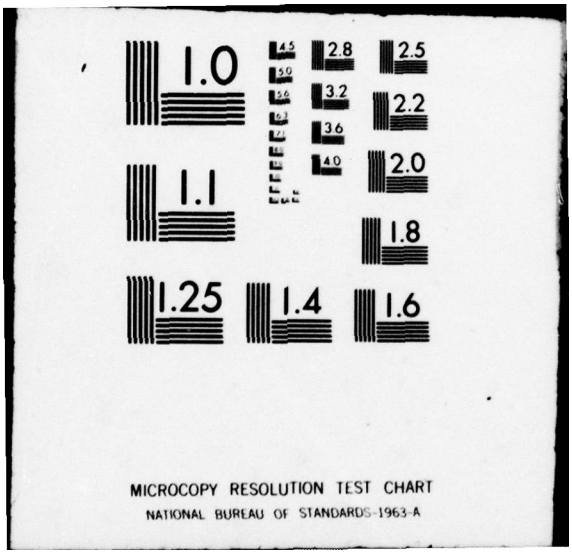
AD-A063 432 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 9/2
NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (N--ETC(U)
SEP 78 C K HILL
UNCLASSIFIED CCTC-CSM-UM-15-78-VOL-3 NL

1 OF 3
AD
A063432


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

COMPUTER SYSTEM MANUAL
CSM UM 15-78
VOLUME III
1 SEPTEMBER 1978

(12)
NW

LEVEL II

C C T C

COMMAND & CONTROL TECHNICAL CENTER

NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (NIPS 360 FFS)

DEFENSE COMMUNICATIONS AGENCY

VOLUME III
FILE MAINTENANCE (FM)

USERS MANUAL

COMMAND AND CONTROL TECHNICAL CENTER

(9) Computer System Manual, (14) Number CCTC-CSM-UM-15-78-VOL-3

(11) 1 September 1978

(6)

NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (NIPS 360 FFS)

Users Manual.

Volume III • File Maintenance (FM).

(12) 220 p.

SUBMITTED BY:

(10) S K Hill

CRAIG K. HILL
Captain USA
CCTC Project Officer

APPROVED BY:

Frederic A. Graf, Jr.

FREDERIC A. GRAF, JR.
Captain U.S. Navy
Deputy Director
NMCS ADP

ACKNOWLEDGMENT

This manual was prepared under the direction of the Chief for Programming with general technical support provided by the International Business Machines Corporation under contracts DCA 100-67-C-0062, DCA 100-69-C-0029, DCA 100-70-C-0031, DCA 100-70-C-0080, DCA 100-71-C-0047, and DCA 100-77-C-0065.

CONTENTS

ABSTRACT

This volume defines the File Maintenance (FM) component of NIPS 360 FPS. It describes the functioning of the component, its capabilities, limitations, expected output results, and specifications for preparing run decks and control cards which will serve as reference for the knowledgeable user.

This document supersedes CSM UM 15-74, Volume III.

CSM UM 15-78, Volume III is part of the following additional NIPS 360 FPS documentation:

CSM UM 15-78	Vol I	- Introduction to File Concepts
	Vol II	- File Structuring (FS)
	Vol IV	- Retrieval and Sort Processor (RASP)
	Vol V	- Output Processor (OP)
	Vol VI	- Terminal Processing (TP)
	Vol VII	- Utility Support (UT)
	Vol VIII	- Job Preparation Manual
	Vol IX	- Error Codes
TR 54-78		- Installation of NIPS 360 FPS
CSM GD 15-78		- General Description

FILE MAINTENANCE (FM)

Section 1

INTRODUCTION

The File Maintenance (FM) volume of the Users Manual is divided into nine sections.

Section 1 presents a brief introduction to the manual.

Section 2 describes the capabilities of the FM component in generating and updating data files.

Section 3 gives detailed information on FM transactions under the control of FMS control card and logic statements.

Section 4 describes input to the FM component.

Section 5 discusses the output from the FM component.

In summary, sections 2 through 5 give the user an insight to the general functioning of the component, its capabilities, and limitations. A thorough understanding of these sections is necessary before attempting to use the system.

The remaining four sections describe the specifications for preparing run decks and control cards for the FM component and serve as a reference for the knowledgable user.

Section 2

FM CAPABILITIES

The FM component provides the NIPS 360 FFS user with a tool for generating and maintaining data files. For maximum efficiency, updating is performed on indexed sequential access method (ISAM) or virtual storage access method (VSAM) files residing on Direct Access Storage devices (DASD). This permits random access of only those records that are to be processed on any given run; only records that require processing are retrieved from the file. Record subsets, whose data content is modified, are written back into the file. The file is updated in place; only those records requiring modification are rewritten.

The component is capable of generating and adding new records and subsets to the file or deleting records and subsets. Another feature of the system is its ability to increase or decrease the size of existing records and subsets through the addition or deletion of variable data. All of these functions can be accomplished in one update cycle.

The component also has the capability to generate and maintain segments of a sequentially organized file. This capability will allow large chronological files to be segmented as specified by the user to expedite processing when the primary updating to be performed is adding new records with a higher key than those already present in the data file. Each segment, when generated, will be a separate data file that can be processed singly or concatenated with other segments as one data file.

During FM, the user has the option of specifying the file block size when generating either an indexed sequential or physical sequential file, or when updating a physical sequential file. If a block size is not specified, the output file block size will be the same as the input FFT or

the input file. For details of file block size specification, see Volume VIII, Job Preparation Manual. Block size modification is not possible with VSAM DATA files during FM

The FM component features are discussed in the following paragraphs. Another option available to the user is to limit the records read from the data file to be processed. This can be done through the use of a LIMIT statement.

2.1 Transaction Sources

The FM component will accept fixed or variable length, blocked or unblocked, or undefined transaction records from tape, disk, or card files. The transaction files may be organized sequentially, or in indexed sequential form on disk. The latter capability permits the FFS data file to be used as transaction input to update another FFS data file. Transaction files may also be VSAM FFS data files.

2.2 Transaction Formats

On any given FM run, file updating can be performed with a variety of transaction record formats, from any number of different reporting sources. This capability allows new information management systems to be developed without seriously impacting existing information management systems and existing reporting systems.

2.3 Logical Updating and Transaction Editing

The FM component automatically performs all of the I/O functions, record positioning, and new record generation required for a file update run; however, the user supplies the actual record update logic to be used by writing file maintenance logic statements. The user writes one logic statement for each different transaction record format that is used to update his file. The logic statements are written in the File Maintenance languages (POOL or NFL). These languages provide a full complement of comparative and arithmetic functions, so that the user can perform all of his transaction data editing validation in conjunction with his file update. In most applications, no preprocessing or preediting of transaction data should be required.

2.4 Data Conversion and Validation - User Subroutines

In addition to the comparative and arithmetic commands, the FM languages also provide the user with commands that allow him to perform on-line validation and conversion of transaction data with subroutines that may be written in any of the S/360 languages (COBOL, FORTRAN, PL1) or in S/360 Assembler language. These subroutines can be prestored on a load library, using the NIPS 360 FPS utility, UTSUBLDR, and can be accessed by the FM component as they are required. A NIPS 360 FPS utility, TABGEN, has been provided to generate tables that perform data conversion and validation using the table search technique (see section 2, Volume VII, Utility Support.)

2.5 Processing of Periodic Sets

The FM languages contain a series of commands that allow the user to scan the periodic sets, so that he may selectively update subsets based on their existing data contents. These commands also allow the user to control the physical sequence of subsets in a set when new subsets are being added.

2.6 Variable Field and Variable Set Maintenance

The FM languages provide a set of commands for adding, deleting, or replacing information in the variable fields and the variable sets.

2.7 Production of Auxiliary Outputs

The FM languages provide a set of commands for producing auxiliary outputs on printer, punch, direct access device or tape in conjunction with a file update. These outputs can be used to provide an audit trail of the updates or to produce transaction files that might be used to update other files in a file system. Two separate printed reports, two separate punched outputs, and up to five auxiliary files may be produced. The user has access to a 999-byte work area for formatting auxiliary output records. Each record is limited to a maximum of 994 bytes because NIPS adds six overhead bytes to the data and the DCB LRECL is 1000. Violation of this limit will cause a System 001 ABEND (I/O ERROR, RECORD TOO LONG). Information can be selected for

output from either the transaction records or the data records. The FM languages also provide the capability for maintaining summary counts and totals which can be output to tape, print, or punch. The user has access to 20 full-word binary work areas for developing summary information in conjunction with a file update run.

2.8 Production of Run History Information

The FM component automatically produces a printed run history to indicate error conditions that were encountered during file updating. The run history is printed separately from the printed auxiliary output.

2.9 Logic Statement Storage

The FM component maintains a library of the user-written logic statements (in compiled form) to update a file. The Logic Statement Library is maintained with the data file and consists of a series of records with special keys. This capability allows the user to avoid recompilation of the statements each time his file is to be updated, thereby reducing execution time. The component automatically retrieves the logic statements that are required for the transaction record formats encountered during an update cycle. The user can add/delete statements on the Logic Statement Library by using a set of control cards, called library action cards (see section 4.1.2).

2.10 File Update Methods

The FM user can cause the FM component to employ any combination of the following file update methods in a given file maintenance run.

a. Range Updating - Using this method, every record of the data file is made available for logical updating. This method may be used to make corrections to some data field in the file or to produce summary information. If at all possible, this method should not be used as a part of normal production runs since it greatly reduces the efficiency of the component. However, when it is deemed necessary, the RANGE statement should be compiled and stored on the library in accordance

with the paragraph "Logic Statements" under section 3.1, Control Elements.

b. Exception Updating - Using this method, each transaction record is matched against a particular data record, which is made available for logical updating. When no match is found, a new data record is generated, and a 'new record' indicator is set which the user can test by using one of the FM language commands in his logic statement. Exception updating is the normal update method.

c. Exception-Range Updating - Using this method, each record that has been subjected to exception updating is made available for further processing by one or more exception-range logic statements. This method can be used for collecting summary information or for producing audit information on all records affected by an update cycle.

d. Direct Subset Updating - Using this method, a transaction record is matched against a particular data record subset and the record's fixed set, and the particular subset is made available for processing. When no subset match is found, a new subset is generated, unless the record does not exist. In the latter case, the transaction is logged as an error on the run history.

2.11 Modes of Operations

Four FM component modes of operation may be specified by the user. (Any given FM run may be executed in only one mode of operation.)

Compile Logic Statement Mode - This mode allows the user to get compilations of his logic statement for the purpose of debugging. The component produces a symbolic listing with all errors flagged, along with a summary of the errors and their causes. Processing is completed when the last statement is compiled. No logic statements can be added to the File Library portion of the data file in this mode, but any other actions can be performed on the Logic Statement Library. If the data file is sequentially organized, no

library actions will be performed on the Logic Statement Library.

Logic Statement Library Update Mode - When this mode of operation is specified, FM updates the Logic Statement Library as specified by the library action cards. Logic statements may also be compiled in this mode and placed on the Logic Statement Library. If the data file is sequentially organized, logic statements cannot be added to the logic statement portion of the data file during this mode.

Data File Generation Mode - The user specifies this mode of operation when he wishes to generate a new file. The component will also compile logic statements and perform Logic Statement Library maintenance if specified. At this time, the user may specify the file block size using the BSZNEWF symbolic parameter of the XFM procedure. If a block size is not specified, it will be set at the size of the input FFT.

Data File Update Mode - This mode of operation is specified when the user wishes to update a data file. Logic statement compilation and library maintenance may also be performed by the component when executed in this mode of operation. If the file is sequentially organized, the user may specify a new file block size using the BSZNEWF symbolic parameter in the XFM procedure. If a new block size is not specified, it will be set at the size of the input file.

2.12 Transaction Sorting

The FM component automatically sorts all transaction records into sequence on the transaction control fields prior to updating a file. The sort will be accomplished on tape or disk. When the transaction volume is such that the disk sort capacity is exceeded, a tape sort will be executed. If the transactions are already in order, the sort is automatically bypassed.

2.13 Ordinary Maintenance

Through the use of the Ordinary Maintenance (OM) Transaction Descriptor (TD) cards (see section 6.2), the

user may specify automatic validation of transaction data, and automatic (unconditional) updating of file fields. The types of data validation that may be performed are value, range, picture, and verification against a table or subroutine. Data conversion through use of table or subroutine is also permitted. Error options are provided to permit or suppress automatic logging of erroneous data, to automatically delete records of subsets generated by the erroneous transaction data, or to automatically clear the data field that corresponds to the erroneous transaction field.

The Ordinary Maintenance capability permits the user to write logic statements that contain only Ordinary Maintenance Transaction Descriptor cards, or to write statements that contain both Ordinary Maintenance cards and FM statements. Instructions are provided in the FM language to interrogate the results of the Ordinary Maintenance validation (see section 7.4.5). This allows the user to perform part of the validation function using the Ordinary Maintenance language, and to perform the more complex edits and data manipulation in the FM language in one logic statement. In a mixed logic statement, the Ordinary Maintenance functions are executed prior to the FM language functions.

2.14 Checkpoint/Restart

During the generate/update phases of SAM processing, the user may invoke the OS/360 checkpoint/restart capability to record timed or end-of-volume checkpoints. End-of-volume checkpoints will be taken on the SAM data file input for an FM SAM update and on the user's SAM transaction input, if there is one, for a generate or an update. The checkpoint/restart capability should be used only during long-running jobs using the execute only procedures. (Note that OS/360 step restart is program-independent and is not the topic of this discussion. A detailed description of the OS/360 checkpoint/restart capability, which is utilized in NIPS, is available to the interested user in IBM Systems Reference Library, Number C28-6708.) A detailed description of how to use checkpoint/restart in NIPS is included in the Job Preparation Manual, Volume VIII.

2.15 Segmented Files

The segmented file capability is primarily for users with large chronological files where the updating process consists of adding new records with higher Record IDs than those already present on the data file. By segmenting the data file to cover a specified range of Record IDs, the user can generate a segment using the FPT and logic statements from the previous segment. The component will build segment control records containing the boundary of a segment and the volume serial number of the segment such that the most recent segment will contain segment control records identifying all previous segments.

The capability exists to allow the user to maintain an ISAM or VSAM data file containing the FPT, logic statements and segment records for his segmented data file. This file would contain the most recent version of all logic statements and segment records identifying all segments used with the capability. In general the ISAM or VSAM data set may be used in a generate segment run in the same manner in which a SAM data set is generated from a DISK FPT. In an update run, the ISAM or VSAM data set must be defined on the ISAMWORK DD card.

Examples are provided in Volume VIII, Job Preparation Manual, to illustrate methods for generating and maintaining segments of a data file.

2.16 Secondary Indexing

Secondary Indexing, if active for the file being updated, operates within FM in an entirely automatic manner, without user intervention. The maintenance function consists essentially of analyzing the updates of the data file and modifying the Index Data Set to conform to the changed data file. The analysis includes all pertinent keyword functions: recovery of words from keyword fields being updated and the application of stop word tables and dictionaries (if any) to the recovered words.

2.17 Auxiliary File Reference

With the POOL language operator, AFR, FM features a capability to retrieve data from the fixed sets of other NIPS ISAM or VSAM files during the update process of FM enabling the user to expand transaction editing without additional tables or multiple passes of transactions against related files. This capability allows the user to retrieve data from an unlimited number of NIPS ISAM or VSAM files other than the primary data file in a single logic statement. The user must supply a DD card for each auxiliary file referenced. OPENs and CLOSEs of the files are automatic with a maximum of five files OPENed simultaneously.

2.18 Logic Statement Size

Neither POOL, OM nor NFL compilers restrict the size of a user written logic statement due to base register limitations. Certain restrictions on logic statement size do exist, however, and are listed below.

a. The user may define no more than 4096 characters of constants or literals in one logic statement, except in an OM/POOL logic statement, where the OM section and POOL section may each contain 4096 bytes of constants or literal values.

b. In POOL and NFL, no more than 4096 bytes of executable code can be generated between user-defined labels. As a practical example, less than 4096 bytes of code are generated by 100 contiguous MAL instructions or 15 contiguous MVP instructions. If a sequence of code in POOL or NFL does generate over 4096 bytes of code without labels present, an assembly addressability error will occur, in which case the user need only label his source instructions at appropriate intervals and recompile the logic statement.

c. For any one logic statement, a maximum of 255 records can be written to the logic statement library, thus limiting a logic statement to a maximum size of approximately 80,000 bytes.

Section 3

FM DESCRIPTION

In addition to the FM logic statements, the FM component functions under the control of information that exists in the user-supplied control cards and in the transaction and data file records. The function and interaction of these elements is discussed in the following paragraphs.

3.1 Control Elements

The FM component operates under the control of the following elements of information: FMS control card, LIMIT control card transactions, and logic statements.

a. FMS Control Card - The user must provide an FMS control card (see subsection 6.1.1 and 6.3.1) for each FM run. It specifies the functions that the FM is to perform.

   - LIMIT Control Card - The user may optionally provide a LIMIT card (see subsection 4.1.1.1 and 6.1.1.1). It specifies the range of records to be processed.

b. Transactions - Transaction records are source-data records containing information used to update the data records. The transaction records may also contain two elements of control information. They are the transaction ID fields and the transaction control fields.

   - Transaction ID Fields - Since more than one type of transaction record can be used to update a given data file, each transaction record must contain information that uniquely identifies its format. The fields that contain this identifying information are the

transaction ID fields. Each transaction may contain up to three ID fields. The aggregate length of the transaction ID fields may not exceed six characters.

- Transaction Control Fields - If the data in a transaction record is to be applied to a particular data file record or a record subset, that transaction must contain control fields. The most significant information contained in these fields is identical to the information contained in the control fields of the data record to which the transaction applies. The least significant information in these fields may be used to control the sequence in which different transactions are processed against a given data record and must not exceed 10 characters. The user specifies the location of transaction control fields through the use of transaction description cards, which are a part of the logic statements. A transaction record may contain up to 255 characters of control information which may include up to 10 characters of user control information in as many as 60 noncontiguous fields.

- Transaction Report Identification - Each transaction that is used to update a file contains a transaction ID. The location of the transaction ID fields for all of the different transactions within a given report must be the same. To identify the location of the transaction ID fields, the user must assign a report name to each set of transactions with ID fields in unique locations. The report name must conform to the system name rules specified in Volume I, Introduction to File Concepts. Before FM will compile the logic statements that are used in processing a given set of transactions, the transaction report identification record must be placed on the Logic Statement Library through the use of a library action card.

Transactions that are being processed during an FM run must be identified by a report name before they can be processed through the use of the FMS control card and report identification card. Transactions from several different reports may be used to update a file on a given run. Report identification cards/records must be inserted in the input stream between groups of transactions from different reports.

c. Logic Statements - In addition to specifying the processing logic, the analyst must also include certain control information in his logic statements. This information is used to specify the type of logic statement he is constructing and to identify the logic statement so that it may be retrieved from the library.

There are six types of logic statements:

- RANGE Statements without Transaction Data - RANGE statements without transaction data are used to perform logical processing on every record in the data file. The analyst specifies this type of statement by omitting transaction information when he makes the library action cards for his logic statement. This type of statement cannot be stored and must be recompiled for each FM execution. It should be noted that this type of statement can easily be converted to a range with transaction data in accordance with the paragraph below to permit storing on the library. A single transaction description card containing a noncontrol field (i.e., $LSNAM,1,1) with the appropriate ASP card would fulfill the requirements for storage of a logic statement. A single transaction with the applicable transaction control ID columns filled in would subsequently invoke the logic statement at generate/update time (e.g., an A in the first position of a transaction).

- RANGE Statements with Transaction Data - RANGE statements with transaction data are used to perform logical processing on every record in the data file and to update the record. The analyst specifies this type of statement by not specifying transaction control fields when he makes up transaction description cards for his logic statement. This type of statement may be stored on the Logic Statement Library and need not be recompiled for later use.

  RANGE statements apply to the entire data file while EXCEPTION statements apply to one specific record. These types of updating operations may be combined.

- EXCEPTION Statements - EXCEPTION statements always require transaction data and are used to process a particular record of a data file. The transaction control fields indicate which record is to be processed. The analyst specifies this type of statement by specifying the location of the transaction control fields in the transaction description cards. EXCEPTION statements may be stored on the Logic Statement Library and need not be recompiled for later use.

- SUBSET EXCEPTION Statements - SUBSET EXCEPTION statements process a particular subset record of a data file. The analyst specifies this type of statement by specifying a subset field as a minor control field in his transaction description cards. The subset ID must be unique within its set. The transaction control fields, major and minor, indicate the subset to be processed. This type of statement may be stored and need not be recompiled for later use.

- EXCEPTION RANGE Statements - EXCEPTION RANGE statements may be with or without transaction data. The statements are used on a per-record basis to process all data records in the file that have been updated via EXCEPTION

statements during a given run. The analyst specifies this type of statement by using the POOL XNP instruction written in the POOL language logic statement, or by punching XNP as the sixth parameter in a free format Add Statement card (subsection 6.1.2) when using Ordinary Maintenance or the NFL. These statements can be stored on the Logic Statement Library when they are associated with transaction data.

- o Logic Statement Identification - All logic statements except RANGE and EXCEPTION RANGE statements without transaction data, must be uniquely identified within a data file. Statement identification is accomplished through the use of the library action card that must precede each logic statement compiled. The report name and the statement must be specified.

The report name identifies the report type and provides the location of the transaction ID field for a set of transaction records; this is the higher level of identification.

The statement name must be identical to the transaction ID on the transactions used with the logic statement. This is the lower level of identification and must be unique within a given report.

3.2 FM Functioning

The FM component is divided into the following functional sections:

a. Initialization - This section processes the user's punched card input. It uses the FMS control card to determine which FM functions are to be performed and sets up a run communication record to control processing for the run. It will also process any segment control cards and update the segment records on the data file. It constructs a disk work file from the user's Logic Statement Library

update cards; if there are any card transactions, these are output to a second work file.

b. Logic Statement Compilation and Library Maintenance - The functions of this section consist of deleting specified reports and statements from the Logic Statement Library, adding new reports to the library, and compiling and adding new statements to the library for a given file. The report information records and the logic statements are maintained on the library in sequence by report and statement name. Each logic statement on the library contains two parts. The first part is a logic statement control record that is used during transaction processing. This record indicates the statement type. If it is an EXCEPTION or SUBSET EXCEPTION statement, it indicates the location of the transaction control fields in the transaction record it processes. The second part consists of the Executable Load module that is produced when the POOL language or NFL statements are compiled.

In addition to updating the Logic Statement Library, this section also produces a listing of the user's Logic Statement Library update deck, with any errors flagged. When the compile-only mode of FM is specified, the Logic Statement Library is only updated by the addition or the deletion of reports/statements. For compiled statements, only the error listing is produced.

c. Transaction Processing - The Transaction Processor matches each transaction record with its appropriate logic statement and creates an update record for each transaction that is used by the File Processing section to perform the actual record update.

After the Transaction Processor reads a transaction record, it extracts the transaction ID fields from the record. The locations of the transaction ID fields are associated with the transaction report name on the Logic Statement Library. It then uses the transaction report name and the transaction ID

to retrieve the control record of the logic statement that will be used to process the transaction.

An update record, containing the name of the logic statement that processed the transaction, a sort key, and the transaction record, are then produced. The high-order byte of the sort key contains the update record type indicator; 'H' for Range updates, 'E' for Exception Range updates, and 'P' for Exception and Direct Subset updates. The collating values of the record-type indicators establish the sequence in which the update records will be passed to the File Processing section, but do not establish the sequence in which the update records will actually be processed. For Exception update records, the sort key will contain only the major ID, one byte of binary zeros for the set number and user control information. For direct subset updates, the sort key will contain the major ID, the set number and the subset control field. The remainder of the sort field will be padded with binary zeros.

At the completion of transaction processing, the update records are ordered on the sort key, if they are not already in sequence.

If the data file is a segmented file and segment processing is to be performed, the update record sort keys are checked to determine if the record key is within the boundaries of the segment being processed. If it is not, an error message will be printed and the update record will be bypassed.

d. File Processing - The Range and the Exception Range update records are read and saved in core. Then processing of the Exception and Direct Subset update records begins. Direct Subset updating for a given record will be performed following the exception updates on the record. If only Direct Subset updating is to be performed on the record, the fixed set and the subset are retrieved, and the updating is performed. If no matching fixed set is found, an error message is logged. When a matching

subset is not found, a new subset is generated, and a 'NEW RECORD' switch is set that can be tested by the user with the POOL 'BNR' instruction or the NFL new record test.

If Range updates are to be made, the entire data file is passed sequentially. As each data file record is read, its record key is matched against the current update record's sort key. If a match is found, the appropriate logic statement is executed, and the current data record is updated with the information in the current update record. A new update record is then read and processing continues on the same data record, until an update record is read with a different sort key. At this point, Range processing is executed against the current record.

If the current update record's sort key has a lower value than the current data record's key, a new record is generated, and the new record switch is set.

If the current data record's key has a lower value than the current update record's sort key, this indicates that no exception processing is to be performed against the data record, and only the range processing is performed.

When Range processing is performed against the records that were not updated by exception processing, the Exception Range logic statements are not executed.

When all of the data records and update records have been processed, the RANGE statements and EXCEPTION RANGE statements that have been collecting summary information are executed once more, to allow them to output the information.

When no Range processing is required, the File Processing Section retrieves and processes only those records with keys that match the Exception update record sort keys. When no match is found, a new record is generated.

File generation is accomplished by generating a new record for each Exception update record with a unique sort key.

Subset Exception updates can be processed during file generation for those records generated by Exception update records.

For sequential processing, the file is passed sequentially as for Range updating. The FFT and logic statement records are copied onto a new sequential output data file. The records are read and updated, and the records whose major IDs have been changed are written on a temporary hold file. This hold file is sorted and merged with the output data file to produce the new data file. If the sequential file is a segmented file, a listing of the segment control records on the data file will be printed indicating segment boundaries and the volume serial number of each segment. During logic statement execution, if a major control field change occurs, the new Record ID is checked to insure that the new ID is within the segment boundary. If the new Record ID is not within the segment boundary, an error message will be printed and the change will not be executed.

A secondary function of the File Processing section is the production of a consolidated auxiliary output file. The records for this file are produced when any of the POOL or NFL instructions that produce printer, punched card, or tape auxiliary output are executed. Control is then passed to the next section.

e. Auxiliary Output Processing - This phase reads the consolidated auxiliary output file, and outputs the individual records to their proper printer, punched card, disk or tape files. When two printer files are requested, they are produced sequentially on the same printer. The two punched card files are punched into pockets 1 and 2 of the card punch.

Section 4

INPUTS

4.1 Card Input

Card input to the FM component consists of the FMS control card, the Logic Statement Library update deck, and the transaction deck.

4.1.1 FMS Control Card

The format of the FMS control card is described in sections 6.1.1 (Free-Format) and 6.3.1 (Fixed-Format).

An FMS control card must be the first card in the FM run deck. It specifies the functions to be performed during the FM run.

4.1.1.1 LIMIT Control Card

The format of the LIMIT control card is described in section 6.1.1.1 (Free Format). The LIMIT control card is optional. When used, it must follow the FMS control card. It specifies the range of records that File Maintenance will process.

4.1.2 Segment Control Cards

The segment control cards will direct the component to perform processing of the segment records on the data file. The segment control cards must appear immediately after the FMS control card. If segmented processing is not desired, these cards must be omitted. The format of these cards is described in section 6.1.2.

4.1.3 Logic Statement Library Update Deck

This deck specifies the maintenance actions to be performed on the Logic Statement Library and consists of the cards discussed. If no Logic Statement Library update is desired on a given run, there is no requirement for this deck.

4.1.3.1 Library Action Cards

The format of the library action cards is discussed in sections 6.1.3 (Free-Format) and 6.3.2 (Fixed-Format).

These cards direct the component to perform a specific type of library update. Each library action card may be used to specify one update action only. The types of updates that may be specified are as follows:

a. Add a Report Name - The component adds the report name and information concerning the report to the library. The name of a report must be added to the library before statements which will be used to process that report can be added to the library. This can be accomplished in one pass if both sets of control cards are in order in the same run deck.

b. Compile and Add a Logic Statement Permanently - The component compiles a logic statement and adds the logic statement to the library.

c. Compile and Add a Logic Statement Temporarily - The component compiles and adds a logic statement to a Temporary Library only for the current run. A statement added temporarily will be the one executed during the run in which it is added, even if another statement with the same name already exists on the Logic Statement Library.

d. Delete All Logic Statements and Report Names For a File - The component deletes from the library all of the report information and logic statements that pertain to the file.

e. Delete a Report - The component deletes the specified report from the library and deletes the

logic statements that are used to process that report. If user wishes to delete more than one report from a file, all $DR cards should be together in the deck.

f. Delete a Statement - The component deletes the specified statement from the library. If user wishes to delete more than one logic statement from a library, all $DS cards should be together in the deck.

Note: Each library action card, which specifies that the component is to compile a statement, must be the first card of the logic statement deck that is to be compiled. All other library action cards must be placed in front of the logic statement decks.

4.1.3.2 Logic Statement Source Decks

Each logic statement source deck consists of the cards that are required to compile and, if desired, add one logic statement to the library. A maximum of 24 logic statement decks may be compiled and added to the library, either temporarily or permanently, during an FM run. There is no limit on the total number of statements that may be used with a file or report.

Each logic statement may consist of the following card types:

a. Library Action (LA) Card - In addition to directing FM to compile and add a statement to the library, this card also provides information about the transaction data that is to be processed with the statement to be compiled. If there is no transaction data to be processed by the statement, the statement is not added to the library, but is compiled only for use with the run in which it is compiled. The format of the LA card is discussed in sections 6.1.3 (Free-Format) and 6.3.2 (Fixed-Format).

b. Transaction Descriptor (TD) Cards - TD cards are used to describe the transaction data which the

statement is to process. They provide the user with the capability of labeling transaction fields so that he may reference the fields by labels in language source statements. These mnemonics need not be unique from file field mnemonics.

The TD cards are also used to specify the locations of the transaction control fields. If transaction control fields are not specified, the component assumes that a Range statement is being compiled.

The format of the TD card is discussed in sections 6.1.3 (Free-Format) and 6.3.3 (Fixed-Format).

c. Language Identifier Card - This card must precede the POOL and NFL source statements. The format of the language identifier card is described in section 6.1.4. This card tells the compiler which source language is being used.

d. Language Source Statement Card - Section 7 discusses the format and logical capabilities of the POOL language, and section 9 discusses the format and logical capabilities of the NFL. The source statement cards specify the processing logic used by the component.

e. Statement END Card - The last card in each logic statement deck must be a statement END card. The format of this card is discussed in section 6.1.5.

4.1.3.3 Logic Statement Library Update Terminator Card

The last card of the Logic Statement Library update deck must be a terminator card if card transactions follow the deck. The format of this card is described in section 6.1.7. This card is required only if library updates are to be performed.

4.2 Transactions

The transaction file can be either cards, tape, or disk (sequential or indexed sequential or VSAM data set) or a combination of multiple sources with the varying data attributes. If it exists soley on cards or is a combination

of more than one source, it must follow the Logic Statement Library update terminator card. The transactions may be of more than one report type for an FM run. Normally, the report type used for a run is entered on the FM control card as a parameter. However, for runs consisting of more than one report type, the transactions applicable to the different reports must be batched separately and each batch must be preceded by a report identifier card. When utilizing more than a single transaction source in an FM run, a report identifier card is required for each source in order to provide FM with the transaction DD name. The format of the report identifier card is discussed in section 6.1.8. Detailed information concerning processing of a NIPS 360 FFS data base as transaction input is contained in the appendix of this manual.

Transaction records must follow the following conventions:

a. Each transaction record may contain one variable field which must be the last field in the record.

b. The maximum size of the transaction record is 1,000 characters.

c. The transaction records must be in one of the standard S/360 formats; fixed or variable length, blocked or unblocked, or undefined. If the record format of the transaction is either variable or undefined, there will be a 4-byte field prefixed to the beginning of the transaction. This field must be accounted for in the user's logic statement, i.e., if the logic statement name starts in position 1 of a fixed transaction, then it will start in position 5 of a variable or undefined transaction (assuming the same transaction,

d. A transaction field that is used by the logic statement as an indirect address must contain a valid field/group name from the file format table of the file to be processed. The transaction may contain more than one indirect address field, however, each field must contain a valid FFT field/group name.

Numeric transaction data will be processed in two different ways, depending on the use to be made of it. (1) If the data is to be moved to a binary area (that is, a binary data file field or a binary work area) or if it is the operand of a numeric instruction (i.e., add, divide, multiply, subtract, or compare numeric operands), the data will be right-justified (i.e., trailing blanks will be deleted) and edited. Editing means that all characters between the + and - signs (if present) and the last non-blank character of the data will be checked to make sure they are numeric. The last character will automatically be considered numeric if it can be converted to binary. (2) If the data is to be moved to a decimal area (a decimal file field or the EBCDIC work area), a check will be made for a + sign or - sign. The sign will be replaced in the receiving field by an EBCDIC zero; a - sign will cause the low-order zone of the receiving field to be made minus. The data will not be right-justified or edited in this case. The numeric portion of each byte will be transferred to the receiving area, and the zone portion of each byte except the last will be set to a hexadecimal 'F'. The low-order zone will always be set to a hexadecimal 'D' (i.e., minus) if the operand contained a minus (-) sign; otherwise, the low-order zone will be transferred. The zone of a low-order blank will be set to hexadecimal 'C' (i.e., plus).

4.3 Subroutine Library

The Subroutine Library must contain all subroutines and tables which will be used during an FM run. The library may be stored on the same volume as the data file and may have a name of the user's choice.

4.4 Data File

The FM component processes indexed sequential data files on disk and sequential data files on tape or direct access devices. The FM component will also process Virtual Storage Access Method (VSAM) data files on the S/370. When desired, the utility program UTBLDISM (see Volume VII, Utility Support) may be used to load sequential data files onto direct access storage.

Section 5

OUTPUTS

5.1 Output Data File

The output from the FM component is a direct access data file, updated in place on disk (ISAM or VSAM processing) or a sequential data file on tape or disk (SAM processing), or a segment of a sequential data file on tape or disk.

5.2 Auxiliary Output

In addition to producing an updated data file, the FM component provides the user with the capability of producing auxiliary outputs under the control of logic statements. Auxiliary output may be produced on disk, magnetic tape, punched cards, or the printer, according to the user's formatting specifications. The output length is limited to a maximum of 994 bytes. NIPS adds six overhead bytes to the output data and the DCB LRECL is 1000. Violation of this limit will cause a System 001 ABEND (I/O ERROR, RECORD TOO LONG).

5.2.1 Tape and Disk Auxiliary Output

During an FM run, the user may produce auxiliary output on five tape or disk files.

5.2.2 Punched Card Auxiliary Output

The FM component provides the user with two punched outputs. Any formatting of punched output must be accomplished by the user in his logic statements. If the user requests that more than 80 characters of data be punched, the data in excess of 80 characters will be punched on subsequent cards.

5.2.3 Printed Auxiliary Output

The FM component provides the user with two printed outputs. The user is responsible for formatting the print lines. The FM component will handle the printing of 132 characters or less. If the user requests that more than 132 characters of data be printed, the data in excess of 132 characters will be printed on subsequent lines.

5.3 Run History

As part of its functioning, the FM component automatically generates a run history on the printer. This includes listings of logic statements that are compiled, and messages indicating that errors, or unusual conditions that might be interpreted as errors, have been encountered during processing.

If segmented file processing is being performed, the segment control records on the current segment will be printed showing the segment boundary and the volume serial number of each segment.

5.4 File Analysis and Run Optimization Statistics

The File Analysis Statistics capability in the FM component provides transactions showing the number of times each logic statement is executed during an FM execution. The data set name (DSNAME) of this data set must be the data file name suffixed by a T. The T is added to ISAM and VSAM names; the S is replaced by T in SAM names. To obtain transaction output, the DSNAME must be cataloged and the user must specify the volume serial (VTRANS) and unit (UTRANS) in the execution procedure. The volume may be any direct access volume.

If the transaction data set exists at execution time, transactions will be added (DISP=MOD). If the data set does not exist, a 5-track data set will be dynamically allocated. The user may change the allocation value by overriding the TRANST DD card space parameter. Transactions are written as fixed length, unblocked, 50-byte records. The format (fixed) and length (50) cannot be changed but the user may change the blocking factor by specifying a DCB BLKSIZE in the TRANST DD card which is a multiple of 50.

If the user specifies a DSNAME (TRANS) in the TRANS DD card, he must supply all parameters required to process the data set. These parameters must conform to the requirements defined above.

The Run Optimization Statistics capability provides the user with statistical data reflecting the core allocation during FM execution. The breakdown of the statistics detail the amount of core used for user subroutines and tables, logic statements, process blocks, I/O buffers, and access methods. It also includes the number of BLDL entries allocated and used and the number of entries required for each subroutine, table, and logic statement to reside in core. The amount of core required for each subroutine, table, and logic statement to reside in core is also output. If subroutines, tables, and logic statements are rolled, this information will be output with the causes for the rolling and the number of times it occurred.

In addition, the user is able to enter override parameters for the number of BLDL entries to allocate, and the size of the processing block desired for storage of the data records during FM processing.

The statistics gathering is initiated through parameters entered in the PARM field of the EXEC card. The parameters and their functions are as follows:

ROS - Indicates that run optimization information is to be gathered and output.

NOROS - Indicates that optimization processing is to be omitted. If no other parameters are coded, this parameter should be omitted as it is the default.

The parameters the user may supply to tailor his core allocation are listed below. Using these parameters, the Run Optimization Statistics are gathered and output unless the NOROS parameter is used.

TCP=NK - The N indicates the number of bytes requested for the process block in 1000 (K) bytes.

TCB=n - The n indicates the number of entries to be used in the BLDL list for SUBSUP, the subroutine supervisor.

TCS - This parameter indicates that the statistics record on the ISAM or VSAM data file is to be used to compute the process block size. This parameter must not be used with TCP and vice versa.

For a more detailed description of the capability see Introduction to File Concepts, Volume I.

Section 6

CONTROL CARD FORMATS

6.1 Free-Format Specifications

This section specifies the preparation requirements for all FM control cards. These cards may be punched in free format or fixed format (see sections 6.1 and 6.3 respectively).

The general rules that apply to free-format control cards are as follows:

a. The control card data must always be punched starting in column 1. The first character of a control card must always be a dollar sign ($).

b. The information in the cards must be punched in a specified parameter sequence.

c. The control card fields must be separated by commas, with no intervening blanks.

d. If the analyst has no requirement for a certain parameter, he must so indicate by punching a comma for that field, except when he has no more fields to punch.

The four control cards that may be formatted in this manner are as follows:

a. The FMS control card

b. The LIMIT control card

c. The Library Action card

d. The Transaction Descriptor card.

The specifications for each of these cards, together with a description of the editing procedures that will be performed by the system, follow. Any errors detected while editing these cards will cause a no-go switch to be set. However, all control cards will be edited before any run which has erroneous control cards is aborted.

6.1.1 FMS Control Card (Free-Format)

Description:

Field 1 - $FMS/AAA-$Card Identifier And Run Mode.

$FMS/ is the card identifier. AAA indicates the run mode and may be one of the following:

COM - Logic statement compilation mode. The input logic statements will be compiled and check list generated. No library actions can be performed

LIB - Logic Statement Library update mode. Logic statements can be compiled, and all library actions can be performed in this mode.

Note: For sequential files, a run mode of COM or LIB will cause only compilation to be performed. No library action will be accomplished.

GEN - Data file generation mode

UPD - Data file update mode.

Note: Logic statements may be compiled and added to the Logic Statement Library with file generation or file update.

Field 2 - File Name

This field must contain a 1- to 7-character file name. The first character must be alphabetic, and no embedded special characters may occur. On a SAM run or an ISAM or VSAM GEN run, this field supplies the name for the new file.

The following parameters (fields 3-6) are only used if the GEN or UPD run modes are specified. If the COM or LIB modes are specified, the remaining fields should be omitted.

Field 3 - Report Name (Optional).

This parameter may be one to seven characters in length and must be alphabetic. It provides FM with the name of the first transaction report that will be processed in the run. This parameter may be omitted if there are no transactions to process, or if the report name for the first set of transactions is to be specified using the report identifier card. This parameter must be omitted if the transactions being processed originate from multiple transaction sources as information as to the source must be supplied through NEW REPORT control card parameters.

Field 4 - Logic Statement Library Update Indicator (Required for UPD or GEN when logic statements are to be compiled on-line).

'LS' indicates the library is to be updated. This parameter is omitted if no update is required.

Field 5 - Data File Type

For GEN mode, the data file type parameter on the FMS control card should be supplied. If it is not supplied, a default option is used.

For UPD mode, the data file type parameter on the FMS control card may be omitted. The default option will be used if the parameter is not supplied.

FILE MAINTENANCE (FM)

The data file type parameters are:

TAPE - For sequential processing (SAM)

DISK - For direct access processing (ISAM or VSAM)

The default option is to process, according to the organization of the input data file which contains the FFT, logic statements and, for UPD runs, data records.

Field 6 - Transaction Source (Required for UPD or GEN Mode)

TAPE - Sequential transactions; file on either tape or direct access storage.

DISK - Transaction source is in indexed sequential organization

SAM - Transaction source is a NIPS 360 FPS ISAM or VSAM data file in sequential organization

ISAM - Transaction source is a NIPS 360 FPS data file in index sequential organization

CARD - CARD must be specified or this parameter omitted entirely when utilizing multiple transaction sources since the NEW REPORT card describing the source must be included in the run deck.

NONE - No transactions

If this parameter is omitted, CARD is assumed.

Field 7 - Segmented File Processing Indicator

SEG - Segmented processing to be performed in this run

NOSEG - Bypass segmented file processing in this run

If this parameter is omitted, the default option is to perform segmented file processing if the data file is a segment and to ignore segmented file processing if the file is not a segment.

6.1.1.1 LIMIT Control Card

Description:

Field 1 - $LIMIT - card identifier

Field 2 - Field name or Group name [m/n] [#SUBTAB]

The field or group name specified must include the high-order character(s) of the major control field. The user has the option to specify partial field notation for the field or group by indicating m/n. This specifies which portions of the record key will be used for comparison. This partial field must start at the first character; i.e. 1/n. In addition, the field or group may be modified by a subroutine expression. Double pound signs (##) suppress automatic table conversion, and the name of the subroutine enclosed in pound signs forces table conversion.

Field 3 - Relational Operator

The relational operators shown below are allowed in the statement formed by the LIMIT operator and condition the selection of records as follows:

EQ - process when equal to
LT - process when less than
LE - process when less than or equal to
GT - process when greater than
GE - process when equal to or greater than
BT - process when equal to or between

The logical connector NOT may precede all relational operators.

Field 4 - Literal(s)

If the BT relational operator is specified, then two literals are required and must be separated by a slash; i.e., a/b.

6.1.2 Segment Control Cards

The segment control cards are used to update the segment records on a segmented data file. The options allowed are as follows:

SEG - This option indicates to the component that the output of a GEN run is to be a segmented data file. This option must be used only when the mode of the run is GEN.

ADD - This option indicates to the component that a new segment record is to be added to the segmented data file. This option may be used in either GEN or UPD mode.

REP - This option indicates to the component that the volume serial number of a specified segment is to be replaced by a new volume serial number.

DEL - This option indicates to the component that the segment record specified by the low key value is to be deleted from the segmented data file.

Note: The actions specified by the segment control cards will be performed at FM initialization time. Therefore, if no logic statements are to be compiled, omit the logic statement parameter on the FMS control card.

The control cards are free format and possible operands are as follows:

$SEG LOKEY HIKEY

$ADD LOKEY HIKEY VOLID

$REP LOKEY OVOLID NVOLID

$DEL LOKEY

where the first parameter is as shown -

LOKEY - the lowest major control field for the segment

HIKEY - the highest major control field for the segment

VOLID - the volume serial number

OVOLID - old volume serial number for REP action

NVOLID - new volume serial number for REP action

The parameters on the segment control cards must be separated by a comma and/or one or more blanks. If the major control field contains special characters or blanks, the field must be enclosed in quotes or at signs. More than one card may be used to specify the action to be performed. However, the action parameter must be the first entry on the first card of a set of cards if more than one card is needed.

Columns 1-71 are used to contain the segment parameter, column 72 is used as the continuation column, and columns 73-80 are ignored.

If the major control field parameter is too long for one card, a nonblank character in column 72 will indicate the continuation of the field in column 1 of the next card. Fields specified in this fashion must have a single quote (') or at sign (@) at the beginning of the field and a single quote or at sign at the end of the field. A maximum of four cards may be used in specifying one field. Continuation indicators should be used only if the field continues through 71 and the beginning of the next card.

6.1.3 Library Action Card (Free-Format)

Description:

Field 1 - Action Identifier (Required)

This field must contain one of the following:

$DF	delete all reports and statements for a file
$AR	add a report name
$DR	delete a report name and all statements for the report
$DS	delete a statement
$ASP	add a statement permanently
$AST	add a statement temporarily.

For a $DF card, all that is needed is the action identifier-$DF.

Field 2 - Report Name

This field is used to specify a report name. The report name may be from one to seven characters long. It is required for all action cards that deal with reports or statements. This is the last field that need be specified for cards with the function code $DR.

Fields 3,4,5 - Position of Transaction Identification Fields (Add Report)

The Add Report card is treated as a special case. Following the report name, it will contain from one to three parameters used to specify the location of transaction identification fields.

The format of this parameter is:

HH-LL

where

H is the high-order position of the transaction ID (or a portion thereof), and L is the low-order position of the transaction ID. The high- and low-order positions may be specified as 1- or 2-digit numbers with a range between 1 and 99. A one-position statement ID field may be specified by one number only.

Example:

$AR,PORT,1-3,80,33-34

This card requests that the system add information about report type PORT to the library. Transactions within report type PORT will contain identification in three fields. Field 1 will be in columns 1-3, field 2 in column 8Ø, and field 3 in columns 33-34.

The following fields are used only with statement action cards: $AST,$ASP,$DS.

Field 3 - Statement Name (Required)

The name may be from one to six characters long. No embedded blanks will be permitted.

If the library action specified in the card is $DS, field 3 is the last field in the card. The following fields pertain only to the add statement cards.

Field 4 - Length of Fixed Transaction Data (Required)

This field may be from one to four digits long and must be a number between 1 and 1ØØØ inclusive.

Field 5 - High-Order Position of Transaction Variable Field (Optional)

If no variable field exists, this parameter may be omitted. This parameter may be from one to four characters long and must be a number between 1 and 1ØØØ inclusive. The value of this parameter must be greater than the value of field 4.

Field 6 - Exception Range Indicator (OM and NFL only)

This field contains 'XNP' if the logic statement being compiled is an Exception Range statement. If the logic statement is not an Exception Range statement, this parameter must be omitted.

Field 7 - NOP Instruction Count

This field contains 'NCT' and signals to the processor program that the test for logic loops should not be performed during the execution of the logic statement.

6.1.4 Transaction Descriptor (TD) Cards (Free-Format)

These cards are provided to allow the user to label transaction fields in writing his logic statements. He may then reference the transaction field label in his logic statements by the label preceded by a dollar sign ($). The format of these cards is as follows:

Field 1 - Field Label (Required)

The field label can be from one to seven characters long and must be preceded by a dollar sign ($). Both alphabetic and numeric characters may be used, but the first character of a label must be alphabetic. No special characters may be used.

Field 2 - High-Order Position of Field in Transaction (Required)

This field may be from one to four digits. If the analyst is using the TD card to assign a label to the variable field of the transaction, this parameter will be the last parameter on the card. In this case, FM will make certain that the position specified in this card coincides with the high-order position of the variable field as it is specified in the library action card.

Field 3 - Low-order Position of Transaction Field (Required Only for Fixed-Length Fields)

This field may be from one to four digits. FM checks to make certain that the value specified here is not greater than the length of the fixed field as specified in the library action card.

Field 4 - Major or User Control Field Designation (Optional)

The field must be punched as a C followed by a 1- to 2-digit number between 1 and 60. The number indicates the sequence in which the transaction control fields must be arranged in order to compare them to the data record ID. The transaction control fields that are assigned the lower sequence numbers and that have an aggregate length equal to the length of the data file major record control group constitute the major transaction control fields. The transaction control fields that are assigned the higher sequence numbers constitute the user control fields; they are not used in matching the transaction to the file record but are used in sorting to control the record update sequence. For example:

```
$RECID,4,10,C1
```

This example describes the transaction field which the analyst wishes to refer to as RECID in his logic statement. It is located in positions 4-10 of the transaction, and it is the major portion of the transaction that corresponds to the data record ID.

Field 5 - Data Mode of the Transaction Field (Optional)

A - Indicates the field contains alphabetic data
B - Indicates the field contains binary data
C - Indicates the field contains coordinate data in internal format
D - Indicates the field contains alphameric (EBCDIC) data.

If this parameter is omitted, D is assumed.

Field 6 - Control Indicator (Optional)

This parameter specifies the type of control field. The following codes may be used.

M - Major control field
S - Secondary control field
N - No control field.

If blank, other parameters on the card are tested and the control field indicator is set by the program.

Field 7 - Corresponding Subset Control Field Mnemonic

This parameter specifies the subset field to be used as a control field on a direct subset update statement. This field should be blank for other types of update statements.

6.1.5 Language Identifier Card

This card will contain either POOL in card columns 16-19 to indicate that POOL language source statements follow or NFL in any three consecutive columns to indicate that NFL source statements follow.

6.1.6 Logic Statement END Card

The END card must be placed at the end of each POOL language logic statement. This card will contain the word END in columns 16-18.

6.1.7 Logic Statement Library Update Terminator Card

This card will contain the word STOP in card columns 16-19. The STOP card must be the last card of the logic statement update deck if card transactions follow the deck.

6.1.8 Report Identifier Card

This card is used to signal the beginning of another report. The characters NEW REPORT will be punched in

columns 6-15. The report name will begin in column 21 and may be from one to seven characters.

If processing transactions from multiple sources, the DD name for the specific transaction source must follow the report name. A comma must separate the report name from the DD name. The DD name must be in the form PSTRANxx for sequentially organized transaction data and ISTRANxx for defining a NIPS ISAM or VSAM file as a transaction source. The xx may be a user coded unique ID for each DD statement. The statements must be included by the user immediately preceding the FM.SYSIN DD * at run execution. Comments may be included in the NEW REPORT identifier card following the DD name after allowing a blank to separate the two. NEW REPORT statements containing the optional DD name parameter may be contained only in the SYSIN data set.

6.2 Ordinary Maintenance (OM) Transaction Descriptor (TD) Cards

The FMS control card and library action control cards must be coded as shown in sections 6.1, 6.1.1, and 6.2.1. OM Transaction Descriptor (TD) cards are free formatted (there is no fixed format capability). OM TD cards may not be used with the New File Maintenance Language (NFL). The parameters consist of keywords followed by single operands or operand lists. Allowable keywords are as follows:

```
FIELD
CONTROL
PICTURE
VALUE
RANGE
VERIFY
CONVERT
GENERATE
ERROR
```

With the exception of the FIELD and CONTROL parameters, the parameters may appear in any sequence. The keywords and operands may be separated by any number of blanks, commas or equal signs. Operands that contain any of these characters must be enclosed in single quotes ('). Operands may not contain quotes in any position. The parameters for a single

transaction field may be entered on any number of cards. The description of each given field may begin anywhere; however, keywords and operands may not be split across input card boundaries. The number of characters per keyword operand may not exceed 52. The total number of characters per keyword operand list may not exceed 900.

Each of the keyword parameters is discussed in the following subsection. The discussion provides a description of each keyword's function, the legal abbreviation for the keyword, and a description of any restrictions on the use of the keyword.

6.2.1 Keyword: FIELD

Abbreviation: FLD

Example:
```
FLD ALPHA 8 11 A
FLD NUMBER 12 15 D
```

Function:

This keyword is used to identify the start of a transaction field description operand list. It must be the first parameter on a card. The FIELD operands must be entered in a specific sequence as follows:

a. User-assigned transaction field mnemonic. This may be a 1- to 7-byte alphameric operand, the first byte of which must be alphabetic. When the GENERATE parameter is used, the transaction field mnemonic must be identical to the mnemonic for the field that will receive the data.

b. High-order position of the transaction field. This operand may be from one to four bytes long and must be numeric.

c. Low-order position of the transaction field (optional). This operand is omitted for variable length transaction fields. The operand may be from 1- to 4-numeric bytes in length.

d. Field notation indicator (optional). This field consists of a 1-byte code to indicate the type of data contained in the transaction field. The legal codes and their meanings are as follows:

'A' - The field contains alphameric data, including blanks and special characters.

'B' - The field contains numeric data in binary format.

'C' - The field contains coordinate data in internal form.

'D' - The field contains numeric data in decimal format.

The default option for this code is 'D'. Decimal ('D') transaction data may be processed in the POOL language by either the arithmetic (MNU, MNC, CON) or logical (MAL, MAC, COA) instructions.

6.2.2 Keyword: CONTROL

Abbreviation: CTL

Example: CONTROL PSCTL

Function:

This keyword is used to specify that a transaction field is a control field. The operand may be a 1- to 2-digit number, to indicate that the transaction field is a major or record control field; or it may be a subset control field mnemonic, which indicates that the transaction field is a subset control field for a direct subset update statement.

For main or record control fields, the number indicates the sequence in which the transaction fields are to be arranged in order to compare them to the data record ID. The transaction fields assigned the lower sequence numbers that have an aggregate length equal to the length of the major data record control group constitute the major transaction control fields; the remaining transaction

control fields are considered user control fields, and are used only to control the update sequence on a given record.

When the control parameter is specified, it must immediately follow the field parameter list.

6.2.3 Keyword: PICTURE

Abbreviation: PIC

Example: PIC AABB B*S (A)NN A(AB)N

Function:

This keyword is used to indicate that character or profile checks are to be performed on a transaction field. The keyword should be followed by masks depicting the types of EBCDIC characters permitted in the defined transaction field. The PICTURE parameter may only be specified for alphameric or decimal transaction fields. Character checks that may be specified by the masks are Alphabetic (A), Numeric (N), Special (S), Blank (B), Nonblank (X), Non-special (Y), and no check or universal match (*).

A direct test for specific characters, as opposed to types of characters, may be specified with a VALUE check, or by enclosing the specific characters in parentheses. The picture masks, not counting parentheses, must be either shorter than or equal to the length of the transaction field defined. If a mask is shorter, only the leftmost characters of the transaction field will be checked, up to the length of the mask. The picture mask is terminated by the occurrence of a blank. Up to 10 masks may be specified in a PICTURE parameter list.

6.2.4 Keyword: VALUE

Abbreviation: VAL

Example: VAL 32768 8192 ***58 'AAbX3'

Function:

This keyword indicates that the transaction field data must match one of the values specified in the operand list that follows the keyword.

The VALUE parameter may only be used in editing alphameric or decimal (Type 'A' or 'D') transaction fields. The values specified in the list will be compared against the transaction data by left-justifying the list values and padding with blanks as required to achieve field length. Value operands may not exceed the transaction field length.

List values for either alphameric or decimal fields may contain any number of letters. If any of the list values contain embedded blanks (b) or commas (,), they must be enclosed in single quotes ('). If any of the list values contain any other special characters, they need not be enclosed in quotes, but the mode specification of the field must be alphameric (type 'A'). A universal match character (*) may be used to indicate positions of a transaction field which are not to be validated.

Up to 10 values may be specified in a VAL parameter list. Validation against more than ten values should be performed with a table or subroutine and the VERIFY keyword.

6.2.5 Keyword: RANGE

Abbreviation: RAN

Example: RANGE ØØ1 499 -1Ø 29

Function:

This keyword is followed by a list of alphameric value pairs and indicates that the transaction field value must fall within the range of at least one of the specified value pairs. This type of editing may not be performed on coordinate (type 'C') transaction fields.

All value pairs must be alphabetic or numeric. A minus (-) sign may precede a value to indicate that it is

negative. The first value of a pair must be less than the second value.

For binary and zoned decimal fields (types 'B' or 'D'), an arithmetic range test is performed. RANGE operands for these field types may not exceed 10 characters and must contain numbers, or numbers preceded by a minus sign.

A logical range check is performed on alphameric fields.

Up to 10 range pairs may be specified in a RANGE operand list.

6.2.6 Keyword: VERIFY

Abbreviation: VER

Example: VERIFY SUBRS

Function:

This keyword will be followed by the name of a single user-supplied validation subroutine or table. It indicates that the field identified by the TD statement is to be passed against the data validation subroutine or table. An indication of the result (valid/invalid) is to be returned to the system.

Validation or conversion subroutines and tables should be written in accordance with the Users Manual, Volume I, Introduction to File Concepts, and Volume VII, Utility Support.

6.2.7 Keyword: CONVERT

Abbreviation: CON

Example: CON TABID

Function:

This keyword will be followed by the name of a single user-supplied data conversion subroutine or table and

indicates that the transaction field identified by the TD card is to be passed through the conversion routine. The result is stored in the original transaction field, left-justified, with trailing blanks to the length of the original field. The result length must be equal to or shorter than that of the original transaction field. A validity indicator will be returned to the system.

Ground rules for writing conversion subroutines and tables are the same as those specified for the VERIFY parameter.

6.2.8 Keyword: GENERATE

Abbreviation: GEN

Example: FIELD FLDNAM 5 1Ø A GENERATE

Function:

This keyword does not expect or require any operands. It specifies that the system should automatically move the data from the transaction field to the data field. The user name specified as the transaction field mnemonic (reference the FIELD parameter) must be identical to the mnemonic assigned to the file field to be updated.

GENERATE may be used to cause updating of existing periodic subsets, when it is used with direct subset updating logic statements. In direct subset updating logic statements, the user specifies the location of the subset control fields in the transaction record by using the control parameters on the appropriate TD cards. If no file match is found for the subset control fields, a new subset is generated and the new record switch is turned 'on' (unless the fixed set does not exist, in which case an error message will be logged). The user may interrogate this switch by the POOL 'BNR' instruction if desired.

This keyword may also be used to create new periodic subsets, when used in exception logic statements or range logic statements, provided that the periodic subsets do not contain user-defined control fields. Each execution of the logic statement will cause generation of new subsets at the

end of each set to be updated. When there is no file match against the major record control field, a new fixed set is automatically generated. The new record switch is set 'on' and may be tested by the POOL 'BNR' instruction, if desired.

The GEN keyword need not be coded for fields identified as control fields.

6.2.9 Keyword: ERROR

Abbreviation: ERR

Example:
```
ERR T
ERR DS
```

Function:

This keyword expects a single parameter to identify the error option to be taken when invalid data is detected in a field by any of the edit functions specified by the TD parameters. The options and their code identifiers are as follows:

CODE	OPTION
D	Data is not to be moved to the data file. The data move instruction created by the GEN parameter for the transaction field will not be executed.
B	Blank or clear the receiving data file field. The data field is set to its null value (blanks for alphameric fields, zeros for numeric and coordinate fields). The option may not be used with record control fields.
*	Accept the data.
T	Delete the transaction. When this option is used, no updating will take place with any of the transaction fields. If a new record or subset has been created by the transaction, it will be deleted.

If this ERR keyword is omitted, the default option is ERR D. If the ERR keyword is specified, it must be followed by a valid code identifier. Omission of the code identifier will result in error message 20036.

As transaction data errors are detected in processing specified by Ordinary Maintenance TD cards, they will be collected and reported as an additional auxiliary output stream during FM execution. The error log format will be as follows:

MAJOR KEY (30 char. max.)	FIELD ID	ERROR MESSAGES (50 char. max.)	DATA (30 char. max.)	ERROR ACTION CODE
xxxxxxxxxxxxxxx	xxxxxx	xxxxxxxxxxxxxx	xxxx	x
xxxxxxxxxxxxxxx	xxxxxx	xxxxxxxxxxxxxx	xxxx	x

The standard error messages generated indicate the data validation failure as specified by the OM keyword parameter. Messages are as follows:

RANGE	ERROR
VALUE	ERROR
PICTURE	ERROR
VERIFY	ERROR
CONVERT	ERROR

The log may be suppressed for a given field by coding an 'S' following the error option code.

Tests for validity of a given field or the entire transaction may be made by the POOL 'BPV', 'BPN', 'BTN', 'BTV' instructions (see section 7.4.5). User-defined error messages for this log require use of the POOL 'ERR' instruction (see section 7.4.6).

The Ordinary Maintenance error log capability may also be used when writing logic statements for the Source Data Automation (SODA) on-line maintenance capability. When used on-line, erroneous data is logged by underlining the erroneous transaction data with a key to the message indicating the type of error detected. (See Terminal

Processing (TP) Component, Volume VI of the NIPS 360 FFS Users Manual.

6.3 Fixed-Format Specifications

The FM component provides the user with the option of preparing the FM control cards in fixed-format, so that compatibility may be maintained with the 1410 NIPS. This format is somewhat more rigid than the free-format in that the data fields must be placed in certain card columns and numeric values must be made a specified length by punching leading zeros.

One format or the other must be used exclusively for any given run. Fixed-format control cards can not be used with Ordinary Maintenance.

6.3.1 FMS Control Card

Format:

```
0    1  1    2       3   4  5
6    0  6    1       0   5  0
AAAA BBB CCCCC DDDDDD  EEE FF GGGG
```

Description:

Field A - Card identification (must contain FMS/).

Field B - Run mode (must contain one of the following):

COM - Compile mode. The input logic statements will be compiled only and a check list generated. No statements can be added to the library in this mode. However, any other library actions can be performed.

LIB - Logic Statement Library update mode. Logic statements can be compiled, and all library actions can be performed in this mode.

FILE MAINTENANCE (FM)

Note: For sequential files, a run mode of COM or LIB will cause only compilation to be performed. No library action will be accomplished.

GEN - Data file generate mode.

UPD - Data file update mode.

Note: Logic statements may be compiled and added to the Logic Statement Library with file generation or file update.

Field C - File name. The file for which all processing specified is being performed. A file name must be specified in this field.

The following fields (D through G) are left blank if LIB or COM mode is selected.

Field D - This field is used to specify the report name for the first set of transactions that will be processed. The report name may be from one to six characters long and must be left-justified. A report name need not be specified in this field if the user can use the report identifier card. However, when the user is updating with tape transactions, it is quite often more convenient to specify the report name here than to try to generate a report identifier card image on his transaction tape.

Field E - This field is used to specify the transaction source.

SIU - Card transactions.

MRC - Tape transactions.

BLANK - No transaction data.

Field F - LS - Indicates that logic statement compilation or Logic Statement Library

maintenance is to be performed during the run.

BLANK - No logic statement compilation or Logic Statement Library maintenance is to be performed.

Field G - DISK - For direct access processing (ISAM or VSAM).

TAPE - For sequential processing (SAM).

6.3.2 Library Action Cards

Format:

```
Ø    Ø  1 1  11   22   2 3    3   4    4
1    6  Ø 2  56   Ø1   5 Ø    5   Ø    5
     MNNNN OOOOAAAAACCCCC DDDDDD   LLLLLL

     5 5   5          6
     Ø 2   5          7
     BBB   EEFFGGHHIIJJK
```

Description:

Field A - Card Identification. It must contain the word START.

Field B - Action Identification. It must contain one of the following codes, left-justified:

DF - Delete all reports and all statements for a file.

AR - Add a report name.

DR - Delete a report name and all statements for a report.

ASP - Add a statement permanently.

AST - Add a statement temporarily

DS - Delete a statement.

Field C - File Name. It must contain a 5-character alphabetic file name. If the library action card is being used to add a file, delete a file, or request compilation of a Range statement without transaction data, only fields A through C need to be punched.

Field D - Report Name. This field contains a 1- to 6-character report name, left-justified. If the library action card is being used to delete a report, field D is the last field that need be punched.

Fields E through K are required only when the action code is Add Report (AR). They are used to specify the number and location of the transaction identifier fields. At least one and up to three identifier fields may be specified in this manner. Each location is specified by a pair of 2-digit numbers. The total length of transaction ID fields may not exceed six characters.

Field E (Required) - NN - High-order location of the major identification field.

Field F (Required) - NN - Low-order location of the major identification field.

Field G (Optional) - NN - High-order location of the second identification field.

Field H (Optional) - NN - Low-order location of the second identification field.

Field I (Optional) - NN - High-order location of the third identification field.

Field J (Optional) - NN - Low-order location of the third identification field.

Field K (Required) - N - Number of transaction ID fields. N must be a number between one and three.

FILE MAINTENANCE (FM)

The following fields are used only when compiling, deleting, or adding a statement to the library.

Field L - Statement Name. This field contains a 1- to 6-character statement identifier, left-justified.

Field M - This field must contain a T, indicating that the logic statement will be used with transaction data.

Field N - NNNN-Length of the fixed portion of the transaction data. NNNN must be a 4-digit number not greater than 1,000 with leading zeros as required.

Field O - NNNN-High-order position of the transaction's variable field. NNNN must be a 4-digit number with leading zeros as required.

BLANK - No variable transaction field.

6.3.3 Transaction Descriptor (TD) Cards

Format:

```
Ø Ø     1 1   2 2  2 3  333 3 4    4   5 5
1 6     2 6   Ø 1  4 Ø  345 7 Ø    5   Ø 2
  BBBBBBB AAAAA CCCC DDDDEE F GGGGGG   H IIIIIII
```

Description:

Field A - Card Identification. It must contain the word FIELD.

Field B - Transaction Field Label. Labels may be from one to seven characters in length and must be left-justified in this field. The following rules apply to transaction field labels.

a. They may be alpha or numeric characters, but they must begin with an alpha character.

b. No embedded blanks may appear in a label.

Field C - NNNN-Transaction Field - High-order position. Must be a 4-digit number, with leading zeros as required. This field may be left blank on a Transaction Descriptor card that is part of a set of cards that are being used to describe contiguous fields. However, if this field is left blank, extreme care must be taken to maintain the sequence of the cards. If the Transaction Descriptor card is being used to assign a label to a variable transaction field, field C is the last one that need be punched. The TD card for the variable field must be the last card in the Transaction Descriptor deck.

Field D - NNNN-Transaction Field Length. Must be a 4-digit number with leading zeros as required.

Field E - N-Transaction Control Field Indication. If the transaction field is part of the control group, N must be a 2-digit number between one (Ø1) and sixty (6Ø). A one indicates that the field contains the major control. The numbers Ø2 through 6Ø are used to indicate the minor fields. If a field is not part of the transaction control, N must be blank or zero (ØØ).

Field F - Mode of Transaction Field. This parameter specifies the mode of the field. The following codes may be used.

A - Alphabetic
B - Binary
C - Coordinate
D - Alphameric

If omitted, D is assumed.

Field G - Statement Name (optional). This field, if used, contains the name of the statement being processed.

Field H - Control Indicator (optional). This field specifies the type of control field, and may contain the following codes.

M - Major
S - Secondary
*N - Not control

*If this parameter is omitted, other parameters are checked and the program sets the control indicator accordingly.

Field I - Corresponding subset field. For a direct subset update statement only. This parameter specifies the subset field to be used for a control field on a direct subset update statement. It must be omitted for all other types of statements.

6.3.4 Language Identifier Card

Format:

```
Ø    1111
1    6789
     AAAA
```

FILE MAINTENANCE (FM)

Description:

Field A - Language Identification.

POOL - Indicates POOL language source cards follow.

Section 7

POOL LANGUAGE

7.1 Card Format

The format of POOL coding is as follows: Each instruction operator is coded in columns 16-18. Operands begin in column 21. If any instruction is labeled, the label will appear as a mnemonic in columns 6-12.

```
Ø     1     2
6     6     1
SYMBOL OPERATOR OPERANDA,OPERANDB,OPERANDC,OPERANDD
```

7.1.1 Symbols

Instruction symbols may be from one to seven characters long. The first character must be alphabetic. The remaining characters may be alphameric.

7.1.2 Operators

The operators specify the logical functions to be performed. These are discussed in detail in section 7.3.

7.1.3 Operands

There may be up to four operands per instruction, depending on the operator. The operands are separated by commas or a single blank. The types of operands and the coding requirements for each type are discussed in section 7.2.

7.1.4 Comments

The user may also comment on his instructions. At least two blank must separate the comment from the last operand.

7.2 Operand Coding

The operands of each POOL instruction generally specify fields, indicators, and tags upon which the POOL operator acts. The following conventions are used in specifying the operands:

a. Data File Fields--Fields of a data file are indicated by the actual field or group name assigned in the File Format Table (FFT). These are denoted by one to seven characters. The first character must be alphabetic, and the field name must not contain embedded special characters.

b. Transaction Input Fields--Transaction input fields (input data used in updating the data base) can be identified by an alphabetic mnemonic from one to seven characters in length, prefixed by a dollar sign ($). The transaction field name must be identified by means of a TD card (refer to sections 6.1.3 and 6.2 or section 6.3.3).

   The transaction field may also be specified by the form Tm/n, where "m" and "n" specify the high- and low-order positions of the transaction field in the transaction record. No spaces are left between the characters of this operand, e.g., $ITEM or T40/55.

c. Indirect Designation--A data file field may be indirectly designated by the contents of the transaction input record. In this case, the letter C prefixes the usual operand form for the transaction field except when high- and low-order transaction field positions are specified. In this case, the C replaces the T, e.g., C$TRAN or C20/27. The contents of the transaction field must be a data file field or group name as specified in the FFT.

d. Subroutine Processing Validity Codes--Three of the POOL macro instructions have an operand which controls the printing of error messages when an incorrect or invalid value is processed by a table/subroutine. When an invalid input is detected, it will not be used by FM. However, if

the value can from a transaction record, the remaining data will be used as specified in the logic statement. The validity codes (C operands) function in the following fashion:

Code A
or R - When the input data is invalid, the data, its source, and an error message are printed on the run history file.

Code I - No printing occurs in event of an error.

e. Instruction Tags--When one of the POOL instructions specifies a branch to another instruction, the operand is the 1- to 7-character alphabetic tag assigned to the branched-to instruction.

f. Literals--Literals may be used as operands in most instructions. The literals may be alphabetic, signed numeric or unsigned numeric. Alphabetic literals are enclosed within single quotes. All enclosed literals are considered alphabetic if any character included is other than a valid numeric character.

Numeric literals do not need to be enclosed within single quotes. An unsigned numeric literal is considered positive, and a signed numeric literal will have the sign set in the low-order byte during processing.

A signed numeric literal enclosed in quotes will cause the sign to be carried as an additional character.

One restriction applies to the use of alphabetic literals; a single quote may not be embedded in a literal.

g. EBCDIC Work Area--FM provides a 999-byte EBCDIC work area for formatting auxiliary output records, for accumulating summary information and for passing information between logic statements. POOL instructions will reference this area by the operand notation WX/Y where X and Y are the high-

and low-order locations of a work area field, e.g., W1/1ØØ or W2ØØ/23Ø. A maximum of 994 bytes may be specified in auxiliary output instructions (e.g., POOL PRT, NPL PRINT). Violation of this limit will result in a System 001 ABEND.

h. Binary Work Area--FM provides a second work area which can be used to perform arithmetic functions in binary. This area will be 2Ø words (8Ø bytes) long. POOL instructions will reference words in this area by the notation BX/ or BXX/ where X must be between one and 20. This area will be restricted to arithmetic computation.

7.3 POOL Instructions

7.3.1 Alphabetical Listing

In this list, all POOL instructions are presented alphabetically with their group numbers.

Group	Instruction	Function
2	ADC A,B,C	A+B=C if A is not blank
2	ADD A,B,C	A+B=C
27	AFR A,B,D,C	Store value at D from field C in NIPS record ID B of file DDNAME A
3	BEQ A	branch if B EQ A
22	BFN A,B	branch to B if invalid OM field A.
22	BFV A,B	branch to B if valid OM field A.
3	BGE A	branch if B GT or EQ A
3	BGT A	branch if B GT A
3	BLE A	branch if B LT or EQ A

FILE MAINTENANCE (FM)

Group	Instruction	Function
3	BLT A	branch if B LT A
3	BNE A	branch if B NOT EQ A
3	BNR A	branch if new record
3	BNV A	branch if invalid (GEN, TBL,VAL,AFT)
3	BPO A	branch if pgm sw 1 is ON
3	BRA A	branch unconditionally
3	BRO A	branch if overflow
9	BSS A	create subset for field A before subset or at end of set if all records have been processed.
3	BS2 A	branch if pgm sw 2 is ON
3	BTN A	branch if OM found any field invalid.
3	BTV A	branch if OM found all fields valid.
13	CLR A	set field A to blanks
17	COA A,B	compare alphabetic B to A
18	CON A,B	compare numeric B to A
20	CVF A	delete variable field A in current periodic subset, or delete all subsets in variable set A.

FILE MAINTENANCE (FM)

Group	Instruction	Function
4	DDR	delete current NIPS record and branch to logic statement exit.
5	DSB A,B	delete subset containing field A. Address the next subset in the set or branch to B if the last subset was deleted.
9	DSC A	delete subset containing field A. Address the next subset.
2	DVC A,B,C	A/B=C if A is not blank
2	DVD A,B,C	A/B=C
4	END	end of logic statement.
23	ERR A,B	SODA: underscore the invalid field A and display the literal B. OM: error log field A with literal B.
6	GEN A,B,C,D	call sub/tab B to process field A. Store the result at D. Use validity code C.
4	HLT	establish logic statement exit
3	LNK A	branch to A. Save NSI address for RET.
10	LOG A	print field A on first printer (same as PRT)
1	MAC A,B	move alphabetic A to B if A is not blank.

FILE MAINTENANCE (FM)

Group	Instruction	Function
1	MAL A,B	move alphabetic A to B
19	MCS A,B	move alphabetic A to minor control field B.
14	MCT A,B	move alphabetic A to major control field B.
14	MCW A,B	execute MCT and write immediate
7	MNC A,B	move numeric A to B if A is not blank
7	MNU A,B	move numeric A to B
2	MUC A,B,C	A*B=C if A is not blank
2	MUL A,B,C	A*B=C
21	MVF A,B	move A to variable field or variable set B. Variable set field length is FFT definition; transaction field may be split
21	MVR A,B	move A to variable field or variable set B. Variable set field length is transaction length
4	NCT	no operation
4	NOP	no operation
25	OVF A,B,C	move variable field A to work area B. Move maximum 256 bytes per execution. When the last segment of the variable field has been moved, branch to C

FILE MAINTENANCE (FM)

Group	Instruction	Function
12	PCH A	punch A (pocket 1)
12	PC2 A	punch A (pocket 2)
5	POS A,B	find first subset for field A. If none, go to B
16	POV A,B,C	find first subset for field B which contains value A. If none, go to C
12	PRT A	print field A on first printer (same as LOG).
12	PT2 A	print field A on second printer.
4	RET	branch to instruction following last LNK.
26	RVF A	reset OVF segment pointer for variable field or variable set A to first byte in the field
11	SDT A	store FM start date/time in A. YYDDDTTTT.
4	SIE	set condition to EQ
4	SIH	set condition to GT
4	SIL	set condition to LT
3	SMR A	branch to A when last RANGE record has been processed.
4	SOF	set overflow OFF
4	SOO	set overflow ON
4	SPF	set pgm sw 1 OFF

FILE MAINTENANCE (FM)

Group	Intruction	Function
24	SSS A	store consecutive number in field A (all subsets).
5	STP A,B	address next subset for field A. If none, resume sequential execution of POOL instructions.
16	STV A,B,C	address subset which follows subset for field B which contains value A. If none, go to C.
2	SUB A,B,C	A-B=C
2	SUB A,B,C	A-B=C if A is not blank.
4	SVF	set validity sw to valid (see SNV).
4	SVO	set validity sw to invalid (see BNV).
4	S2F	set pgm sw 2 OFF
4	S2O	set pgm sw 2 ON
6	TBL A,B,C,D	lookup value A in table B using condition C. Store the function at D.
8	VAL A,B,C	call subroutine B to process value A using condition C.
12	WRT A	write A on auxiliary file 1
12	WT2 A	write A on auxiliary file 2.
4	SPO	set pgm sw 1 ON
4	SRF	set new record sw OFF
4	SRO	set new record sw ON

FILE MAINTENANCE (FM)

Group	Instruction	Function
12	WT3 A	write A on auxiliary file 3.
12	WT4 A	write A on auxiliary file 4.
12	WT5 A	write A on auxiliary file 5.
4	XNP	branch to exit if no exception processing. If not the first instruction in a RANGE statement it is a no-operation.

7.3.2 Valid Operands Chart

Symbols used in the chart have the following meanings:

TF - transaction field
DF - data file field
IA - indirect address
WA - work area
LV - literal value
SL - statement label
ST - subroutine or table
VC - code value
--
A - alphabetic
N - numeric
B - binary
E - EBCDIC(work)
C - coorindate
--
CF - control field
PS - periodic set
GR - group
V - variable field(periodic or variable set)

FILE MAINTENANCE (FM)

Group	TF	DF	IA	WA	LV	SL	ST	VC	A	N	3	P	C	CF	PS	GR	V
OPERAND 'A'																	
1	X	X	X	X	X				X	X		X	X	X	X	X	
2	X	X	X	X	X					X	X	X		X	X		
3						X											
4	none																
5		X	X						X	X	X		X	X	X	X	
6	X	X		X	X				X	X	X	X		X	X	X	
7	X	X	X	X	X					X	X	X		X	X	X	
8	X	X	X	X	X				X	X	X	X		X	X	X	
9		X	X						X	X	X		X	X	X	X	
10	X	X		X	X				X	X		X		X	X	X	
11		X	X	X					X	X	X	X			X	X	
12	X	X		X	X				X	X		X		X	X	X	
13	X	X	X	X					X	X	X	X	X		X	X	
14	X	X	X	X	X				X	X	X	X		X	X	X	
15	reserved																
16	X	X	X	X	X				X	X	X	X		X	X	X	
17	X	X	X	X	X				X	X			X	X	X	X	
18	X	X	X	X	X					X	X	X		X	X	X	
19	X	X	X	X	X				X	X	X	X		X	X	X	

FILE MAINTENANCE (FM)

Group	TF	DF	IA	WA	LV	SL	ST	VC		A	N	B	E	C	CF	PS	GR	V
20		X								X	X	X	X			X	X	
21	X	X	X	X	X					X	X		X		X	X	X	
22	X									X	X	X		X				
23	X									X	X	X	X	X				
24		X	X								X	X			X	X	X	
25		X								X	X			X		X		X
26		X								X	X	X		X	X	X	X	X
27	X			X	X					X	X		X					

FILE MAINTENANCE (FM)

Group	TF	DF	IA	WA	LV	SL	ST	VC	A	N	B	E	C	CF	PS	GR	V
OPERAND 'B' - Omitted groups have no B operand																	
1		X	X	X					X	X		X	X	X	X		
2	X	X	X	X	X					X	X	X		X	X	X	
5						X											
6								X									
7	X	X	X	X	X					X	X	X			X	X	
8						X											
14		X	X						X	X	X			X		X	
16		X	X						X	X	X			X	X	X	
17	X	X	X	X	X				X	X			X	X	X	X	
18	X	X	X	X	X					X	X	X		X	X	X	
19		X	X						X	X	X			X	X	X	
21		X							X	X		X			X	X	
22						X											
23					X				X								
25				X								X					
27	X			X	X				X	X		X					

FILE MAINTENANCE (FM)

Group	TF	DF	IA	WA	LV	SL	ST	VC	A	N	B	E	C	CF	PS	GR	V
OPERAND 'C' - Omitted groups have no 'C' operand																	
2		X	X	X						X	X	X			X	X	
6								X									
8								X									
16						X											
25						X											
27	X			X	X				X	X		X					
OPERAND 'D' - Omitted groups have no 'D' operand																	
6		X		X					X	X	X	X			X	X	
27				X								X					

7.3.3 Instruction Groups

The POOL instructions are organized into various groups according to the number of operands and operand types allowed. Therefore, all the permissible operands are indicated by the respective group of each instruction. The following subsection, which discusses each POOL instruction, will also indicate to which group it belongs.

The following tables are organized by instruction groups. For each group, the number of operands, the instructions that belong to the group, and a list of the legal operands are shown. The following legend should be used to interpret the legal operands.

DF	-	Data File
IA	-	Indirect Address
LV	-	Literal Value
SL	-	Symbolic Instruction Label
ST	-	Subroutine or Table
TF	-	Transaction Field
VC	-	Validity Code
WA	-	Work Area

Group 1

```
*************************************************************
*Number of Operands-2                                       *
*Instructions-MAC,MAL                                       *
*                                                           *
*Legal A Operands                    Legal B Operands       *
*  TF,WA,LV,IA,DF                      WA,IA,DF             *
*************************************************************
```

Group 2

```
*************************************************************
*Number of Operands-3                                       *
*Instructions-ADC,ADD,DVC,DVD,MUC,MUL,SUB,SUC               *
*                                                           *
*Legal A Operands                    Legal B Operands       *
*  TF,WA,LV,IA,DF                      TF,WA,LV,IA,DF       *
*                                                           *
*Legal C Operands                                           *
*  WA,IA,DF                                                 *
*************************************************************
```

FILE MAINTENANCE (FM)

```
Group 3
*****************************************************************
*Number of Operands-1                                           *
*Instructions-BEQ,BGE,BGT,BLE,BLT,BNE,BNR,BTV,BTN,              *
*             BNV,BPO,BRA,BRO,BS2,LNK,SMR                       *
*                                                               *
*Legal A Operands                                               *
*  SL                                                           *
*****************************************************************

Group 4
*****************************************************************
*Number of Operands-None                                        *
*Instructions-DDR,END,HLT,NCT,NOP,RET,SIE,SIH,                  *
*             SIL,SOF,SOO,SPF,SPO,SRF,SRO,SVF,                  *
*             SVO,S2F,S2O,XNP                                   *
*****************************************************************

Group 5
*****************************************************************
*Number of Operands-2                                           *
*Instructions-DSB,POS,STP                                       *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  IA,DF                               SL                       *
*****************************************************************

Group 6
*****************************************************************
*Number of Operands-4                                           *
*Instructions-GEN,TBL                                           *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,DF                         ST                       *
*                                                               *
*Legal C Operands                    Legal D Operands           *
*  VC                                  WA,DF                    *
*****************************************************************
```

FILE MAINTENANCE (FM)

Group 7

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-MNC,MNU                                           *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF                      TF,WA,LV,IA,DF           *
*****************************************************************
```

Group 8

```
*****************************************************************
*Number of Operands-3                                           *
*Instructions-VAL                                               *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF                      ST                       *
*                                                               *
*Legal C Operands                                               *
*  VC                                                           *
*****************************************************************
```

Group 9

```
*****************************************************************
*Number of Operands-1                                           *
*Instructions-BSS,DSC                                           *
*                                                               *
*Legal A Operands                                               *
*  IA,DF                                                        *
*****************************************************************
```

Group 10

```
*****************************************************************
*Number of Operands - 1                                         *
*  Instructions - LOG                                           *
*                                                               *
* Legal A Operands                                              *
*  TF,LV,EBCDIC WA, ALPHA AND DECIMAL DF                        *
*****************************************************************
```

FILE MAINTENANCE (FM)

Group 11

```
*****************************************************************
*Number of Operands-1                                           *
*Instructions-SDT                                               *
*                                                               *
*Legal A Operands                                               *
*  WA,IA,DF                                                     *
*****************************************************************
```

Group 12

```
*****************************************************************
*Number of Operands-1                                           *
*Instructions-PCH,PC2,PRT,PT2,WRT,WT2,WT3,                      *
*             WT4,WT5                                           *
*                                                               *
*Legal A Operands                                               *
*  TF,LV, EBCDIC   WA, Alpha and Decimal DF                     *
*****************************************************************
```

Group 13

```
*****************************************************************
*Number of Operands-1                                           *
*Instructions-CLR                                               *
*                                                               *
*Legal A Operands                                               *
*  TF,WA,IA,DF                                                  *
*****************************************************************
```

Group 14

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-MCT,MCW                                           *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF                      IA,DF                    *
*****************************************************************
```

Group 15 Reserved

FILE MAINTENANCE (FM)

Group 16

```
*****************************************************************
*Number of Operands-3                                           *
*Instructions-POV,STV                                           *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF                      IA,DF                    *
*                                                               *
*Legal C Operands                                               *
*  SL                                                           *
*****************************************************************
```

Group 17

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-COA                                               *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF                      TF,WA,LV,IA,DF           *
*****************************************************************
```

Group 18

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-CON                                               *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF                      TF,WA,LV,IA,DF           *
*****************************************************************
```

Group 19

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-MCS                                               *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF                      IA,DF                    *
*****************************************************************
```

FILE MAINTENANCE (FM)

Group 20

```
*****************************************************************
*Number of Operands-1                                           *
*Instructions-CVF                                               *
*                                                               *
*Legal A Operands                                               *
*  DF                                                           *
*****************************************************************
```

Group 21

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-MVF, MVR                                          *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF,WA,LV,IA,DF  (Variable Field)     DF                      *
   TF              (Variable Set)       DF                      *
*****************************************************************
```

Group 22

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-BFV,BFN                                           *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*  TF                                  SL                       *
*****************************************************************
```

Group 23

```
*****************************************************************
*Number of Operands-2                                           *
*Instructions-ERR                                               *
*                                                               *
*Legal A Operands                    Legal B Operands           *
*   TF                                 LV                       *
*****************************************************************
```

FILE MAINTENANCE (FM)

```
Group 24
*****************************************************************
*Number of Operands - 1                                         *
*Instructions - SSS                                             *
*                                                               *
*Legal A Operands                                               *
*  IA,DF,NUMERIC OR BINARY                                      *
*****************************************************************

Group 25
*****************************************************************
*Number of Operands-3                                           *
*Instructions-OVF                                               *
*                                                               *
*Legal A Operands                  Legal B Operands             *
*  DF                                WA                         *
*                                                               *
*Legal C Operands                                               *
*  SL                                                           *
*****************************************************************

Group 26
*****************************************************************
*Number of Operands-1                                           *
*Instructions-RVF                                               *
*                                                               *
*Legal A Operands                                               *
*  DF                                                           *
*                                                               *
*****************************************************************

Group 27
*****************************************************************
*Number of Operands - 4                                         *
*Instruction - AFR                                              *
*                                                               *
*Legal A Operands                  Legal B Operands             *
*   TF,LV,WA                         TF,LV,WA                  *
*                                                               *
*Legal C Operands                  Legal D Operands             *
*   TF,LV,WA                         WA                        *
*                                                               *
*****************************************************************
```

7.4 POOL Instructions

There are 84 POOL instructions available, each of which can be used to build the updating logic statements for FM. These instructions can be divided into five basic types according to the function they perform. These are:

a. Environment establishing instructions which allow the user to selectively handle the periodic set fields of the data base.

b. Data handling instructions which allow the user to perform processing against the fields of the data base such as moving, arithmetic operations, and transformation.

c. Control instructions used to control the sequence of instruction execution.

d. Display instructions which allow the user to specify certain output functions.

e. Validity test instructions and error logging instructions used with OM and SODA.

Each of the 84 POOL instructions (within its basic type) is described below.

7.4.1 Environment Handling Instructions

Position to First Subset of Periodic Set (Group 5)

POS A,B

This instruction causes the first subset of the periodic set containing field A to be made active; i.e., positioned so that processing can be performed on the data contained in the first subset. If there are no subsets of the set in the current data record, the instruction with tag B will be executed next.

Step to Next Subset of Periodic Set (Group 5)

STP A,B

FILE MAINTENANCE (FM)

This instruction causes the next subset after the current active subset of the periodic set containing field A to become active. If there is no other subset to be made active, the activity is, in effect, placed after the last subset and sequential execution of the POOL instructions resumes. If the instruction is successful the next subset is made active and the instruction with tag B is executed.

Position on Value (Group 16)

POV A,B,C

This instruction causes the activity for a periodic set to be placed at the first subset with the field specified by operand B containing the data value specified by operand A. If no subsets of the set meet the requirement, the instruction with tag C will be executed next.

Step to Value (Group 16)

STV A,B,C

This instruction causes the next subset, after the current subset that contains the data value specified by operand A in the field specified by operand B, to be made active. If no subsets of the set meet the requirements, the instruction with tag C is executed next.

Delete Subset of Periodic Set and Branch (Group 5)

DSB A,B

This instruction causes the currently active subset of the periodic set containing field A to be deleted from the current data record. If the deleted subset was the last of the periodic set, the subset activity is placed after the last one, and a transfer to the instruction with tag B is

made. Otherwise, the next subset is made active, and sequential operation continues.

Delete Subset of Periodic Set and Continue (Group 9)

DSC A

This instruction causes the active subset of the periodic set containing field A to be deleted from the current data record. If there is a subset of the same set following the deleted subset, it is activated. If not, the activity is placed after the last one. In either case, sequential operation continues.

Build Subset of Periodic Set (Group 9)

BSS A

This instruction causes a subset of the periodic set containing field A to be built, and placed in the current data record. The active subset of the periodic set and any that follow are moved down in the data record, and the newly generated subset is placed in the position formerly occupied by the last active subset. If the activity was placed after the last subset of the periodic set by some previous instruction, the new subset is placed after the last one. In either case, the newly created subset becomes active and is ready for processing by other instructions.

Delete Current Data Record (Group 4)

DDR

This instruction notifies FM that the current data record is to be deleted; i.e., not to be included in the output data file. After execution of this instruction, all processing is terminated for the update statement.

Sequence Subsets (Group 24)

SSS A

FILE MAINTENANCE (FM)

This instruction places sequence numbers in field A of the periodic containing field A. The sequence numbers will always begin with one and will always be incremented by one for each subset. This instruction may not be used to modify the contents of the system generated PSSQ fields.

7.4.2 Data Handling Instructions

Auxiliary File Reference (Group 27)

AFR A,B,C,D

This instruction allows the user to move the contents of field C of the data record with major record identification B of the ISAM or VSAM file with DDNAME A into the EBCDIC work area D. The auxiliary file must be a NIPS data file of ISAM or VSAM organization, but must not be the primary data file. Field C must be an alpha field or group, a numeric field or group, a decimal field or group, variable field, or a coordinate field within the fixed set. Binary and coordinate data is converted to EBCDIC before being placed into the work area D. If the data from the field C has an output length greater than the specified work area length truncation occurs from the right. If field C is shorter than the work area, the data is left justified and padded on the right with blanks. If any error occurs that prohibits the return of data to the work area (i.e., no DD card with DDNAME A, no existing major record identification B, I/O error), an error is logged fully describing the problem, and the validity indicator is set to invalid. Only if the referenced data is successfully returned to work area D is the validity indicator set to valid. AFR operates in a read only mode; no data can be entered into the auxiliary file with this instruction.

FILE MAINTENANCE (FM)

Move Alphabetic Field to Field (Group 1)

MAL A,B

This instruction moves the content of field A to field B. The content of field A is treated as alphabetic information. If the field to be moved is shorter than field B, blanks are appended on the right. If the field to be moved is longer than field B, the resultant content of field B is truncated from the right.

Note: The 'MAL' instruction is based on a 360 ALC move instruction which moves left to right through each field a byte at a time. Therefore, caution must be used whenever fields overlap (e.g., MAL W10/12,W11/13).

Move Numeric Field to Field (Group 7)

MNU A,B

This instruction causes the content of field B to be set to the content of field A. The content of field A is assumed to be a numeric quantity. If the field to be moved is shorter than field B, necessary zeros are appended on the left. If the field to be moved is longer, the resultant content of field B will be truncated from the left. Numeric transaction data may be signed or unsigned. Signed transaction data will have the sign set in the low-order byte during processing; if unsigned, the data is considered positive. Transaction data may be signed in two ways: (1) the sign may be placed in the low order zone; e.g., -2 would be punched as a K, +1 would be punched as an A, etc.; (2) a + sign or - sign may immediately precede the first digit of the data (although a + sign will in effect be ignored).

Note: The 'MNU' instruction, when moving data from a decimal area to a decimal area, uses a 360 ALC move instruction which moves left to right through each field a byte at a time. Therefore, caution must be

used whenever fields overlap (e.g., MNU W10/12,W11/13).

Move Conditional Alphabetic Field to Field (Group 1)

MAC A,B

This instruction is executed exactly as MAL except that if the field to be moved is blank, no action is taken; i.e., field B is left undisturbed.

Move Conditional Numeric Field to Field (Group 7)

MNC A,B

This instruction is executed exactly as MNU except that if the content of field A is blank, no action is taken; i.e., all of field B is left undisturbed.

All of the following arithmetic operations use three operands. The user specifies in the A and B operands the fields with which the arithmetic operation is to be performed. These two fields are not disturbed by the operation.

The C operands specify the field into which the result of the operation is to be stored.

If he desires, the user may specify the same field for all three operands or for any two operands. For instance, he may add A to B and store the result in A.

Add Field A to Field B, Store in Field C (Group 2)

ADD A,B,C

This instruction algebraically adds the content of field A to the content of field B and stores the result in field C. The resultant sum is moved numerically to field C, and the sum is truncated or extended by zero characters on the left as appropriate. If the resultant sum is truncated, the overflow indicator is turned on.

FILE MAINTENANCE (FM)

Add Conditional Field A to Field B, Store in Field C (Group 2)

ADC A,B,C

This instruction is identical in execution to ADD except that if the content of field A is blank, no action is taken.

Multiply Field A by Field B, Store in Field C (Group 2)

MUL A,B,C

This instruction causes the content of field A to be algebraically multiplied by the content of field B and the result to be stored in field C. As with ADD the result is stored numerically in field C, and the overflow indicator is set as appropriate.

Multiply Conditional Field A by Field B, Store in Field C (Group 2)

MUC A,B,C

This instruction is identical in execution to MUL except that execution does not take place if the content of field A is blank.

Divide Field B into Field A, Store in Field C (Group 2)

DVD A,B,C

This instruction causes the content of field A to be divided by the content of field B and the result to be stored in field C. Results are moved to field C numerically from right to left, and zeros are truncated or added on the left as required. The overflow indicator is set as in MUL. Remainders are lost.

Divide Conditional Field B into Field A, Store in Field C (Group 2)

DVC A,B,C

FILE MAINTENANCE (FM)

This instruction is executed exactly as DVD except that execution does not take place if the content of field A is blank.

Subtract Field B from Field A, Store in Field C (Group 2)

SUB A,B,C

This instruction algebraically subtracts the content of field B from the content of field A and stores the result in field C. The result is stored numerically from right to left, and zeros are truncated or added to the left as required. If field C will not contain the result, the overflow indicator is turned on. Otherwise, it is turned off.

Subtract Conditional Field B from Field A, Store in Field C (Group 2)

SUC A,B,C

This instruction is identical in execution to SUB except that if the content of field A is blank, the content of field C remains unchanged.

Clear Variable Data Field (Group 20)

CVF A

This instruction causes the variable data field of the current data file record to be cleared. The length of the variable data field is effectively set to zero.

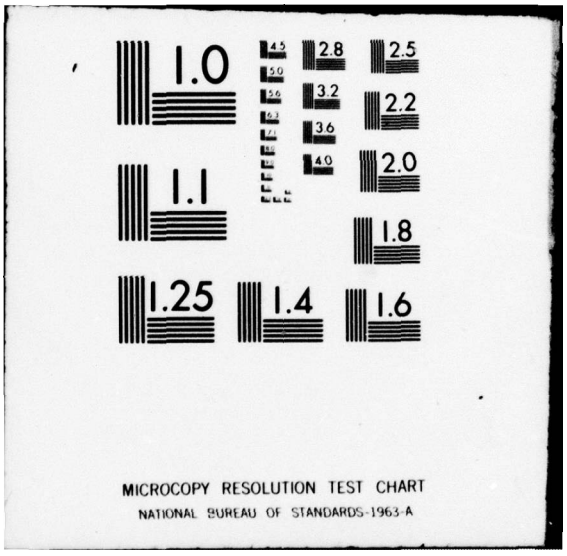
Move to Secondary Control Field (Group 19)

MCS A,B

This instruction moves the contents of field A to field B. The instruction is treated as an MAL instruction. This is the only instruction that may be used to alter the contents of a subset control field.

AD-A063 432 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 9/2
NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (N--ETC(U)
SEP 78 C K HILL
UNCLASSIFIED CCTC-CSM-UM-15-78-VOL-3 NL

2 OF 3
AD
A063432


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

This instruction is executed exactly as DVD except that execution does not take place if the content of field A is blank.

Subtract Field B from Field A, Store in Field C (Group 2)

SUB A,B,C

This instruction algebraically subtracts the content of field B from the content of field A and stores the result in field C. The result is stored numerically from right to left, and zeros are truncated or added to the left as required. If field C will not contain the result, the overflow indicator is turned on. Otherwise, it is turned off.

Subtract Conditional Field B from Field A, Store in Field C (Group 2)

SUC A,B,C

This instruction is identical in execution to SUB except that if the content of field A is blank, the content of field C remains unchanged.

Clear Variable Data Field (Group 20)

CVF A

This instruction causes the variable data field of the current data file record to be cleared. The length of the variable data field is effectively set to zero.

Move to Secondary Control Field (Group 19)

MCS A,B

This instruction moves the contents of field A to field B. The instruction is treated as an MAL instruction. This is the only instruction that may be used to alter the contents of a subset control field.

FILE MAINTENANCE (FM)

Move Control Field (Group 14)

MCT A,B

This instruction moves the contents of field A to field B. If truncation is required, the instruction is treated as the move alphabetic instruction (MAL). MCT and MCW are the only instructions which can alter the contents of the record control field. Subsequent transactions that match the old record key are still applied to this record.

Move Control Field and Write the Data Record (Group 14)

MCW A,B

This instruction functions in the same manner as MCT except that the data record is output upon completion of the execution of the logic statement in which an execution of an MCW instruction has occurred. A subsequent transaction that matches the old record key will cause a new record with the old key to be generated.

If a NIPS ISAM file is being updated and the step abnormally terminates, all records whose control field was changed up to that point have been deleted from the file. The new records may not have been written onto the file, depending on where the step termination occurred.

Look Up Tabular Function Value (Group 6)

TBL A,B,C,D

This instruction causes the contents of the transaction input field A to be used as an argument to the tabular function B (code conversion table) to produce a function value which is stored in field D. The result of the tabular function is assumed to be alphabetic and is left-justified in field D. The function value is truncated or padded with blank characters on the low-order end as required. If the input field A does not compare with entries of the tabular function, the disposition of the transaction input record is determined by C. The legal values of C are A, I, or R. The transaction input record disposition is as described for the C operand of VAL. The validity indicator is set to valid or invalid, as appropriate, by this instruction. Note that inputs to tables and subroutines may be data base fields, work areas, or transaction fields.

Generate Value from Argument Field (Group 6)

GEN A,B,C,D

The execution of this instruction is identical to that of TBL except that B is the name of a non-tabular function.

Store Date-Time in Field (Group 11)

SDT A

This instruction stores the current Date-Time Group (DTG) in field A. The DTG reflects the time that this FM run started. The format of the value stored is XXYYYZZZZ in which

XX = Year (ØØ-99)
YYY = Date (ØØØ-366)
ZZZZ = 24 hour time (ØØØØ-24ØØ) to the hundredths of hours

The 9-character DTG field is stored alphabetically in field A. The DTG is truncated or padded on the right as required (see MAL).

Move/Add Variable Field to Variable Field (Group 21)

MVF A,B

This instruction updates variable fields in periodic sets and adds data to the end of variable sets. When used to update periodic sets, the contents of field B will be replaced by the contents of field A. When used to add to variable sets, the transaction input data (field A) is appended to the variable set identified by field B. The input data is packed into fixed-length subset records; new subset records are created whenever necessary.

Move to Variable Set/Field (Group 21)

MVR A,B

This instruction appends the contents of the transaction input field A to the contents of the variable data field specified by the B field. If the original length of the variable data set is N, and the length of field A is M, the resultant length of the data set will be N+M. If the B field is a variable field in the periodic set, the previous contents of the field will be replaced by the contents of field A. The difference between MVF and MVR is this: the length of each subset field in a variable set is obtained from the FFT by MVF; the lengths of all subset records in the set are the same (except for the last, which may be shorter). MVR uses the input transaction length (field A) as the output length; subset records may vary in length. This method reduces the probability of splitting words between subset records. Only words which are split between input transactions will be split between subset records.

FILE MAINTENANCE (FM)

Output Variable Field to Work Area (Group 25)

OVF A,B,C

This instruction will move a portion of the variable field A to the defined work area specified by the B field. The variable field A may be stored in a periodic set or a variable set. The B field also determines the length of the move. Up to 256 characters can be moved at a single execution. A pointer is then set to the remaining unmoved portion of the variable A field so that the next execution of this instruction will move another portion of the variable field data. If the remaining portion of the field is shorter than the work area receiving the data, the receiving field is first cleared and then the data is moved. The pointer is then reset to the beginning of the variable field, and a branch is taken to the label specified in the C operand. If the length of the variable field is zero, the work area is cleared to blanks and a branch is taken to the C operand label.

Reset Variable Field Pointer to Beginning of the Variable Field (Group 26)

RVF A

This instruction will reset the pointer which is used in the OVF instruction back to the beginning of the variable field. It may be used only when an OVF instruction with the same A operand appears within the logic statement.

Clear Field (Group 13)

CLR A

This instruction sets field A to blanks. Field A cannot be a record control ID.

Note: The processing operators discussed above, with the exception of MVF, MVR, CVF, OVF, and RVF apply both to nonperiodic and periodic fields. Reference may be made to

nonperiodic fields at any point in a logic statement, but reference to fields of a periodic set may occur only when a subset of that periodic set is currectly active. Periodic sets are independent of each other, and subsets of two or more periodic sets may be active simultaneously. Only one subset of a given periodic set may be active at any given time.

7.4.3 Control Instructions

Compare Field to Field Alphabetic (Group 17)

COA A,B

This instruction compares the content of field A to the content of field B. Both operand fields are assumed to be alphabetic, and comparison takes place from left to right. If the B field is shorter than the A field, the comparison will be made as if the B field were padded on the right with blanks. If the A field is shorter than the B field, only the common portion of the two fields is compared. The compare indicator is set as follows:

Condition	Compare Indicator Setting
A equal to B	EQUAL
B greater than A	HIGH
B less than A	LOW

Compare Field to Field Numeric (Group 18)

CON A,B

This instruction causes the content of field B to be compared to the content of field A. Both operand fields are assumed to be numeric and comparison is algebraic. The relative lengths of the operand fields are not a criterion for meaningful comparison. Both operand fields are assumed to be assigned whole numbers. The sign of each operand field is indicated by the setting of

the zone bits of the low-order character. The compare indicator is set as follows:

Condition	Compare Indicator Setting
A equal to B	EQUAL
B greater than A	HIGH
B less than A	LOW

Check Field for Validity (Group 8)

VAL A, B, C,

This instruction causes the contents of field A to be processed by the subroutine B to determine if the content meets specifications contained in subroutine B. Field C specifies whether to log an error if the content of field A is determined to be invalid. The codes for field C are:

Value of C	Disposition or Action
A or R	Appropriate error message is logged on history file and invalid data is skipped
I	No error messages printed - invalid data is skipped

The validity indicator is set to valid or invalid, as appropriate, by this instruction.

Branch Unconditionally (Group 3)

BRA A

This instruction causes an unconditional branch to the instruction with tag A.

Branch Unconditionally and Link (Group 3)

LNK A

This instruction causes an unconditional branch to tag A. The location of the instruction following the LNK instruction is saved for use with the RET instruction.

Return Branch (Group 4)

RET

This instruction causes an unconditional branch to the instruction following an LNK instruction. An LNK instruction must have been executed prior to the execution of the RET instruction. These two instructions can be used to provide closed subroutines in logic statements. Only one level of closed subroutines may be used at any given time.

The following instructions test the compare indicator. The indicator is reset to indicate the result each time a compare instruction is executed.

Branch if not Equal (Group 3)

BNE A

This instruction causes a transfer to the instruction with tag A if the compare indicator is either high or low.

Branch if Greater Than or Equal (Group 3)

BGE A

This instruction causes a transfer to the instruction with tag A if the compare indicator is set either high or equal.

Branch if Less Than (Group 3)

BLT A

This instruction causes a transfer to the instruction with tag A if the compare indicator is low.

FILE MAINTENANCE (FM)

Branch if Equal (Group 3)

BEQ A

This instruction causes a transfer to the instruction with tag A if the compare indicator is equal.

Branch if Less Than or Equal (Group 3)

BLE A

This instruction causes a transfer to the instruction with tag A if the compare indicator is set equal or low.

Branch if Greater Than (Group 3)

BGT A

This instruction causes a transfer to the instruction with tag A if the compare indicator is set high.

Set Indicator Low (Group 4)

SIL

This instruction sets the indicator low.

Set Indicator Equal (Group 4)

SIE

This instruction sets the indicator equal.

Set Indicator High (Group 4)

SIH

This instruction sets the indicator high.

The following instructions are concerned with the validity indicator. Execution of the TBL, GEN, AFR, and VAL instructions sets this indicator to valid (off) or not valid

(on). It is initialized to valid before the execution of each logic statement. During OM POOL logic statement execution, the switch is set to invalid by OM if a picture error occurs.

Branch if Not Valid (Group 3)

BNV A

This instruction causes a transfer to the instruction with tag A if the validity indicator is set to not valid.

Set Validity Switch On (Group 4)

SVO

This instruction sets the validity switch on, (not valid).

Set Validity Switch Off (Group 4)

SVF

This instruction causes the validity switch to be turned off (valid).

The following instructions are concerned with the overflow switch. This switch is set on any time an arithmetic operation results in an overflow condition. It is turned off by any arithmetic operation that does not result in an overflow condition. It is initialized to off before execution of the logic statement begins.

Branch if Overflow (Group 3)

BRO A

This instruction causes a transfer to the instruction with the tag A if the overflow indicator is set to on.

Set Overflow Switch Off (Group 4)

SOF

This instruction causes the overflow switch to be turned off.

Set Overflow Switch On (Group 4)

SOO

This instruction sets the overflow switch on.

The following instructions are concerned with the new record switch which is turned on any time a new record is generated. New records are generated when an equal comparison of the input transaction and data record control fields cannot be made. The new record switch is turned on prior to the execution of the first logic statement used to process a new record. It will be turned off prior to executing any subsequent logic statements that are used to process the same record.

Branch if New Record (Group 3)

BNR A

This instruction causes a transfer to the instruction with tag A if the new record switch is on.

Set New Record Switch On (Group 4)

SRO

This instruction sets the new record switch on.

Set New Record Switch Off (Group 4)

SRF

This instruction causes the new record switch to be turned off.

The following instructions deal with the two program switches. These switches are provided to assist the user in controlling his logic. The user may turn them on and off as required. They are initialized to off prior to each

execution of a logic statement. Note: Each valid transaction will execute a logic statement.

Set Program Switch On (Group 4)

SPO

This instruction sets the program switch on.

Set Program Switch No. 2 On (Group 4)

S2O

This instruction sets program switch No. 2 on.

Set Program Switch Off (Group 4)

SPF

This instruction causes the program switch to be turned off.

Set Program Switch No. 2 Off (Group 4)

S2F

This instruction causes program switch No. 2 to be turned off.

Branch if Program Switch On (Group 3)

BPO A

This instruction causes a transfer to the instruction with tag A if the program switch is on.

Branch if Program Switch No. 2 is On (Group 3)

BS2 A

This instruction causes a transfer to the instruction with tag A if program switch No. 2 is on.

FILE MAINTENANCE (FM)

The following miscellaneous control instructions are also available to the user.

Branch on Summary (Group 3)

SMR A

This instruction causes a transfer to the instruction with tag A when the last record has been processed. This instruction is valid only for range statements.

Halt (Group 4)

HLT

This instruction specifies that no further processing is to be performed against the current data record.

No Operation (Group 4)

NOP

This instruction performs no operation.

End of Logic (Group 4)

END

This instruction signifies to the language compiler that the end of the logic statement has been reached.

NOP The Diagnostic Instruction Counter (Group 4)

NCT

This instruction is a carryover from 1410 NIPS and has been included in NIPS 360 FFS to maintain compatability. The NCT instruction has no operational function in NIPS 360.

Exit if no Prior Processing on Data Record (Group 4)

XNP

This instruction is used to provide the Exception Range capability. It is effective only when it appears as the first instruction in a RANGE statement. If it appears anywhere else, it functions as an NOP. When this instruction appears as the first instruction in a RANGE statement, it causes an immediate exit from the RANGE statement if the data record being processed has not been subjected to Exception updating during the run.

7.4.4 Display Instructions

For all display instructions, the length of field A cannot exceed 994 bytes. NIPS adds six overhead bytes to the data specified by field A and the DCB LRECL is 1000. Violation of this limit will result in a System 001 ABEND (I/O ERROR, RECORD TOO LONG).

Log Comment (Group 10)

LOG A

This instruction causes the contents of field A to be written on the printer in batch mode and on the display screen in TP mode.

Note: Carriage control may be specified by coding an optional B operand. This operand, if coded, must be preceded by a comma and enclosed in quotes. Valid B operands are:

0 (zero)	-	space two lines
- (minus)	-	space three lines
1 (one)	-	eject (In TP mode, this character will cause the screen to blank)

Print Comment (Group 12)

PRT A

FILE MAINTENANCE (FM)

This instruction is the same as log. See above note.

Print Comment (Group 12)

PT2 A

This instruction is the same as LOG and PRT except it writes operand A data on the printer in batch mode and on an OMQ display device in TP mode. See above note.

Write on AOF (Group 12)

WRT A

This instruction causes the contents of field A to be written on the auxiliary output file.

Write on Second AOF (Group 12)

WT2 A

This instruction causes the contents of field A to be written on the second auxiliary output file.

Write on Third AOF (Group 12)

WT3 A

This instruction writes the contents of field A on the third auxiliary output file.

Write on Fourth AOF (Group 12)

WT4 A

This instruction writes the contents of field A on the fourth auxiliary output file.

Write on Fifth AOF (Group 12)

WT5 A

This instruction writes the contents of field A on the fifth auxiliary output file.

Punch Output (Group 12)

PCH A

This instruction causes the contents of field A to be punched in punch pocket one.

Punch Output (Group 12)

PC2 A

This instruction punches the contents of field A in punch pocket two.

7.4.5 Ordinary Maintenance Validity Test Instructions

The following group of instructions may be used to test the results of the Ordinary Maintenance validation functions. Instructions are provided to allow testing of the validity of a given field, or the entire transaction. These instructions are as follows:

Branch on Transaction Field Valid (Group 22)

BFV A,B

Example - BFV $TRANS,OK

This instruction branches to location 'B' if the contents of transaction field A passed the Ordinary Maintenance edit tests. The transaction field must be designated by its assigned mnemonic.

Branch on Transaction Field Not Valid (Group 22)

BFN A,B

Example - BFN $TRANS,BAD

This instruction branches to location 'B' if the contents of transaction field 'A' failed any of the Ordinary

Maintenance error tests. The transaction field must be designated by its assigned TD mnemonic.

Branch on Transaction Valid (Group 3)

```
BTV     A
```

This instruction branches to location 'A' if all of the transaction data passed on specified Ordinary Maintenance validity tests.

Branch on Transaction Not Valid (Group 3)

```
BTN     A
```

This instruction branches to location 'A' if any transaction fields failed the specified validity tests.

7.4.6 Transaction Error Log Instruction (SODA and OM)

Log Erroneous Transaction Data (Group 23)

```
ERR     A,B
```

```
Example -     ERR   $TRANS,'THIS IS BAD DATA'
```

This instruction is provided to assist the terminal operator in correcting erroneous transaction data when using the on-line update capability, Source Data Automation (SODA).

When this instruction is executed during a SODA run, it causes the displayed transaction field 'A' to be underscored with a key to the message provided as the literal in field 'B'. (See the Terminal Processing (TP) component volume of the NIPS 360 FPS Users Manual.)

When this instruction is executed during a batch FM run, it causes the message tc be printed on the Ordinary Maintenance error log (see section 6.2). In a combined Ordinary Maintenance/POOL logic statement, this instruction may be used to replace or supplement the standard OM error messages.

7.5 Logic Statement Examples

The following examples illustrate the setup of the FM run deck for updating the Logic Statement Library, and the use of some of the POOL language instructions. Sections 9.5.3 through 9.5.8 provide equivalent NFL logic statements for the POOL language statements of sections 7.5.3 through 7.5.8. All of the examples pertain to the TEST36Ø file. For a description of this file, see the Introduction to File Concepts volume of the NIPS 36Ø FFS Users Manual.

All of the sample logic statements, with the exception of the Range statement, perform updates with transactions from the report 'RPT36Ø'. The different transaction formats within this report are identified by the letters 'A' through 'G' in column 1 of the transaction.

Sections 7.5.1 through 7.5.8 illustrate the free-format FMS control card, library action cards, and TD cards.

Comments cards are shown in each of the logic statements and explain the functions of the statements.

Section 8.1 provides an equivalent Ordinary Maintenance TD of the POOL statement in section 7.5.6. Section 8.2 shows a combination OM/POOL logic statement.

7.5.1 FMS Control Card

The following FMS control card would be used to execute the 'LIB' mode of FM, to perform updates for the logic statement library for the TEST36Ø file:

```
$FMS/LIB,TEST36Ø
```

7.5.1.1 LIMIT Control Card

The following card would be used when the user wanted to limit a range to all the records whose control field, LCTRL, was equal to 'AAA'.

```
$LIMIT,LCTRL,EQ,AAA          or
$LIMIT,LCTRL,BT,AAA/AAA
```

If all records not equal to 'AAA' were desired, then the control card would be:

```
$LIMIT,LCTRL,NOT,EQ,AAA
```

7.5.2 Library Action Card to Add a Report

The following card would be used to add the report 'RPT36Ø' to the Logic Statement Library. The transaction ID field, for transactions within this report, is located in column 1.

```
$AR,RPT36Ø,1
```

This card could also have been punched as follows:

```
$AR,RPT36Ø,1-1
```

However, since the transaction ID field is only one byte long, the '-1' is not required.

7.5.3 Logic Statement Setup

The following example illustrates the organization of the library action card, the TD cards, the language identifier card, and the POOL instruction cards for an Exception logic statement. The sample statement performs updates with the 'A' transaction format of the report 'RPT36Ø'.

The first card for the statement is the library action card. This card specifies that the statement is to be permanently added to the library. It also specifies that the statement will perform updates with the 'A' transaction of report 'RPT36Ø', and that the fixed data in that transaction format is 8Ø bytes long. The transaction does not contain any variable data.

The TD cards follow the library action card. The first field in each of these cards is used to assign mnemonics to the transaction data fields.

The second and third fields specify the high-order position and the low-order position of the transaction fields.

The fourth field is used to specify that a transaction field is a major or user control field. In the example, $RECID is a major transaction control field, and it corresponds to the data record control group, 'UIC'. $SORT is a user transaction control field. It is not used in matching a transaction record to a data record, but is associated with the record control field to control the file processing sequence.

The fifth field in the TD card indicates the type of data that the transaction fields will contain. The 'A' transaction contains alphabetic (A) data and zoned decimal (D) data only. Insertion of this field is optional, with the default option being 'D'.

The card following the TD card is the language identifier card, and contains the word 'POOL' in columns 16-19.

Comment cards, describing the logic statement's function, follow the language identifier card. The comment cards are identified by an asterisk (*) in column 6.

The logic statement first tests the new record switch by using the BNR instruction. A new record will be generated by FM when no data record can be found with a UIC group that matches the contents the $RECID transaction field, in an 'A' transaction. When this occurs, the instructions at NEWREC will be executed. These instructions format a print line in the EBCDIC work area and print the line. At the completion of this function, the instruction sequence at MOVE is executed. This sequence of instructions moves the data from the transaction fields to the data record fields, using the MAL, MAC, and MNC instructions. The MAL and MAC instructions are used to move the alphabetic data to the file record, and the MNC instruction is used to move the numeric data to the file record.

When the MAC instruction at MOVE is executed, no data transfer will take place if the content of the transaction field $HOME is blank. When the MNC instruction is executed,

no data transfer will take place if the content of the transaction field $PERS is blank.

The instructions that move the transaction data to the coordinate fields DAPT1-4 will automatically convert the data to internal coordinate format.

At the completion of execution of the data move instructions, an SDT instruction is executed. This instruction stores the date/time of the update in the data field LAUD.

A HLT instruction is executed next. This instruction causes an exit from the logic statement.

```
FILE MAINTENANCE (FM)

$ASP,RPT360,A,80
$RECID,2,7,C1,A
$SORT,8,8,C2,A
$HOME,10,10,,A
$ATTACH,12,12,,A
$FUTURE,13,13,,A
$POINT,15,25,,D
$AREA1,27,37,,A
$AREA2,38,48,,A
$AREA3,49,59,,A
$AREA4,60,70,,A
$PERS,73,80,,D
          POCL
    * THIS LOGIC STATEMENT WILL UPDATE THE LOCATION AND DEPLOYMENT
    * AREA OF THE SPECIFIED UNIT.  IF A NEW RECORD IS GENERATED, A
    * MESSAGE WILL BE PRINTED AND THE TRANSACTION FIELDS WILL BE
    * MOVED TO THE DATA FIELDS.
    *
          BNR   NEWREC
    MOVE  MAC   $HOME,HOME
          MAL   $ATTACH,ATACH
          MAL   $FUTURE,FUTU
          MAL   $POINT,POINT
```

FILE MAINTENANCE (FM)

```
          MAL   $AREA1,DAPT1
          MAL   $AREA2,DAPT2
          MAL   $AREA3,DAPT3
          MAL   $AREA4,DAPT4
          MNC   $PERS,PERS
          SDT   LAUD
          HLT
NEWREC    MAL   'NEW RECORD GENERATED.  ID IS - ',W1/30
          MAL   $RECID,W31/36
          PRT   W1/36
          BRA   MOVE
          HLT
          END
```

7.5.4 Use of Data Conversion Subroutines

The following logic statement is used with 'B' transactions. These transactions contain only a major control field in positions 2 through 7.

This logic statement verifies that the CNTRY and ACTIV fields, in selected records, contain valid data.

If a new record is generated by a 'B' transaction, the DDR instruction at 'DELETE' is executed to delete the record. Otherwise, positions 1 to 44 of the work area are cleared to blanks, and the data record's UIC field is moved to the work area. Then the data in CNTRY is converted by the subroutine CTRYS, and the result is stored in the work area. If the data is invalid, asterisks are moved to the work area by the instructions at 'ERR1'. Next the data in ACTIV is converted by the subroutine ACTVS, and the result is stored in the work area. If the data is invalid, asterisks are moved to the work area by the instructions at 'ERR2'. Then the contents of the work area is printed, and the logic statement exits.

Note: Execution of the TBL instructions does not alter the contents of the fields CNTRY or ACTIV.

FILE MAINTENANCE (FM)

```
$ASP,RPT36Ø,B,8Ø
$RECID,2,7,C1,A
          POOL
     *    THIS LOGIC STATEMENT WILL EXTRACT THE COUNTRY CODE AND
     *    THE ACTIVITY CODE FOR SPECIFIED UNITS.
          BNR   DELETE
          CLR   W1/44
          MAL   UIC,W1/6
          TBL   CNTRY,CTRYS,I,W1Ø/24
          BNV   ERR1
  ACTV    TBL   ACTIV,ACTVS,I,W3Ø/44
          BNV   ERR2
  PRT     PRT   W1/44
          HLT
  ERR1    MAL   '****************',W1Ø/24
          BRA   ACTV
  ERR2    MAL   '****************',W3Ø/44
          BRA   PRT
  DELETE  DDR
          HLT
          END
```

7.5.5 Periodic Set Processing

The following examples illustrate two methods for updating periodic subsets. Example 1 uses an Exception update statement to perform the updating of the record. The library action card will add statement 'D' of report type 'RPT36Ø' permanently to the Logic Statement Library. The TD cards assign mnemonics to the transaction fields, with transaction field $RECID containing the major control field of the record to be processed.

The 'POV' instruction will cause the subset of periodic set one to be searched for the data field 'MECLQ' being equal to the transaction field '$MEQPT'. If it is not found, a branch is taken to 'NEW'. If it is found the indicated processing will be done.

At 'NEW', a new subset will be built for Periodic Set 1. The subset control field will be set by the 'MCS' instruction, and the remaining transaction fields will be set in the specified data fields. A message is printed to indicate a new subset was created and the transaction field $MEQPT is printed.

Example 2 uses direct subset update capability. The library action card is the same. The TD card for the transaction field '$MEQPT' is different in that the control parameters on the TD card indicate that the field is to be used as a subset control field. It also carries a data record subset control field parameter name 'MECLQ' which is defined as a subset control group in the FFT. If no subset exists with a total record control group (major control field, set number, and subset control group) equal to the update record control group ($RECID, set number, $MEQPT) a new subset is generated by FM and the total record control group is set in the new subset and the new record indicator is set on. If the subset exists, the subset is made active.

The first instruction tests the new record indicator. If it is on, a branch is taken to 'NEW'. If it is not on, the processing is performed.

At 'NEW', the field is updated, and the messages are printed.

FILE MAINTENANCE (FM)

EXAMPLE 1

```
$ASP, RPT360,D,80
$RECID,2,7,C1,A
$MEQPT,10,22,,A
$NOEQPT,25,27,,D
$ADDCODE,29,29,,A
            POOL
    *     THIS LOGIC STATEMENT WILL SEARCH FOR THE SUBSET
    *     CONTAINING THE EQUIPMENT TYPE AND ADD OR SUBTRACT
    *     THE NUMBER OF ITEMS AS SPECIFIED BY THE ADD CODE. IF
    *     THE SUBSET DOES NOT EXIST, A NEW SUBSET WILL BE BUILT
    *     AND THE FIELDS WILL BE UPDATED.
            POV   $MEQPT,MECLQ,NEW
            COA   $ADDCODE,'A'
            BEQ   ADD
            SUB   MEPSD,$NOEQPT,MEPSD
            HLT
ADD         ADD   MEPSD,$NOEQPT,MEPSD
            HLT
```

FILE MAINTENANCE (FM)

```
    NEW         BSS   MECLQ
                MCS   $MEQPT,MECLQ
                MNU   $NOEQPT,MEPSD
                PRT   'NEW SUBSET CREATED'
                END
```

EXAMPLE 2

```
$ASP,RPT360,D,80
$RECID,2,7,C1,A
$MEQPT,10,22,,A,S,MECLQ
$NOEQPT,25,27,,D
$ADDCODE,29,29,,A
                POOL
    *     THIS STATEMENT WILL PERFORM THE SAME FUNCTION, USING
    *     THE DIRECT SUBSET UPDATE CAPABILITY.  IF THE SUBSET
    *     DOES NOT EXIST A NEW SUBSET WILL BE GENERATED AND THE
    *     SUBSET CONTROL FIELD WILL BE AUTOMATICALLY SET BY FM.
    *
                BNR   NEW
                COA   $ADDCODE,'A'
                BEQ   ADD
                SUB   MEPSD,$NOEQPT,MEPSD
                HLT
```

FILE MAINTENANCE (FM)

```
ADD     ADD  MEPSD,$NOEQPT,MEPSD
        HLT
NEW     MNU  $NOEQPT,MEPSD
        PRT  'NEW SUBSET CREATED'
        PRT  $MEQPT
        END
```

7.5.6 Test for Numeric Data

The following logic statement illustrates a simple method of determining if a transaction field contains invalid numeric data (assuming valid data is a nonzero value within the range ±2,147,483,647). This method is especially applicable when it is not known in exactly what position of the field the data begins and/or ends (e.g., when leading zeros are not required or the right-justification capability of the system is being used). It is also applicable when some of the data may be signed and some may be unsigned. (The PICTURE instruction can be used in Ordinary Maintenance or NFL to perform the editing function when the type of data in each column is known - see sections 8.1, 8.2, and 9.4.6.)

When the logic statement is entered, a CON instruction is executed to compare the operand value with zero. This will cause the transaction data to be edited (see section 4.2). If the transaction field contains a valid nonzero numeric value, the comparison will be <u>not</u> equal; if the transaction field contains invalid data, the comparison will be equal.

FILE MAINTENANCE (FM)

```
$ASP,RPT36Ø,E,8Ø
$TRANS,1,8Ø
$RECID,2,7,C1
$PERSONL,1Ø,15
$READAVG,2Ø,22
$RITNM,25,27
            POOL
    *   THIS LOGIC STATEMENT WILL UPDATE NUMERIC FIELDS IN
    *   THE FIXED SET.  THE INPUT TRANSACTION FIELDS CONTAIN
    *   DECIMAL DATA.  THE DECIMAL DATA IS EDITED TO DETERMINE
    *   IF IT CONTAINS ANY INVALID CHARACTERS.  IF AN ERROR
    *   IS DETECTED, AN ERROR MESSAGE WILL BE PRINTED ON THE
    *   AUXILIARY OUTPUT PRINTER AND THE ERRONEOUS TRANSACTION
    *   WILL BE PRINTED.
    *
            CON   Ø,$PERSONL     TEST IF $PERSONL  NUMERIC
            BEQ   ERR1           BRANCH IF INVALID DATA
            MNU   $PERSONL,PERS
    CHK2    CON   $READAVG,Ø
            BEQ   ERR2
            MNU   $READAVG,READAVG
    CHK3    CON   Ø,RITNM
            BEQ   ERR3
```

FILE MAINTENANCE (FM)

```
            MNU   $RITNM,RITNM
            HLT
ERR1        PRT   'THE $PERSONL FIELD CONTAINS INVALID'
            PRT   'NUMERIC DATA OR A VALUE OF ZERO'
            PRT   $TRANS
            BRA   CHK2
ERR2        PRT   'THE $READAVG FIELD CONTAINS INVALID'
            PRT   'NUMERIC DATA OR A VALUE OF ZERO'
            PRT   $TRANS
            BRA   CHK3
ERR3        PRT   'THE $RITNM FIELD CONTAINS INVALID'
            PRT   'NUMERIC DATA OR A VALUE OF ZERO'
            PRT   $TRANS
            HLT
            END
```

7.5.7 Production of Summary Information

The following logic statement is a Range statement that functions without transaction data. Note that the library action code for this statement contains only the function code '$AST', and that there are no TDD cards for this statement. This type of statement must be compiled on-line each time it is used.

This statement uses the SMR instruction to determine when line processing is completed. When processing is not complete, the three instructions that follow the SMR instruction are executed, and a logic statement exit is taken. These instructions accumulate information in the first three words of the binary work area.

When processing is completed, the SMR instruction causes a branch to the instruction sequence at 'LAST', where final processing and printing of the accumulated data takes place. Note that when the data is moved from the binary work area to the EBCDIC work area, it is automatically converted to zoned decimal format.

FILE MAINTENANCE (FM)

```
$ASI
            POOL
     *      RANGE STATEMENT TO CALCULATE TOTAL NUMBER OF UNITS
     *      IN THE DATA FILE, TOTAL PERSONNEL STRENGTH, AVERAGE
     *      OF TOTAL READINESS AVERAGE OF ALL UNITS
     *      THIS INFORMATION WILL BE PRINTED ON THE AUXILIARY
     *      OUTPUT FILE
     *
                 SMR   LAST
                 ADD   B1/,1,B1/
                 ADD   PERS,B2/,B2/
                 ADD   READAVG,B3/,B3/
                 HLT
     LAST        MAL   'TOTAL NUMBER OF UNITS-',W1/23
                 MNU   B1/,W24/29
                 PRT   ' '                    PRINT BLANK LINES
                 PRT   ' '
                 PRT   W1/29
                 MAL   'TOTAL PERSONNEL STRENGTH OF ALL UNITS-',W1/39
                 MNU   B2/,W40/49
                 PRT   ' '
                 PRT   W1/49
```

FILE MAINTENANCE (FM)

```
        MAL   'AVERAGE OF TOTAL READINESS AVERAGE-',W1/35
        DVD   B3/,B1/,W36/37
        PRT   ' '
        PRT   W1/37
        HLT
        END
```

7.5.8 Variable Field and Set Processing

The following two logic statements illustrate the use of the MVF instruction.

Example 1 moves information from a variable transaction field, $VAR, to the variable field COMMENT. Existing information in COMMENT will be destroyed. When the data transfer takes place, the data is truncated, so that any trailing blanks in the variable transaction field are not moved.

Example 2 appends information to the variable set REFER. Since the information to be transferred is in a fixed length transaction field, no truncation takes place.

Example 3 outputs variable field (REMARK) data. When a matching subset is found, the data is moved to the work area in 50 character increments, and printed.

FILE MAINTENANCE (FM)

EXAMPLE 1

```
$ASP,RPT36Ø,F,1Ø,11
$RECID,2,7,C1,A
$VAR,11
              POOL
    *
    *    THIS LOGIC STATEMENT REPLACES THE INFORMATION IN THE
    *    VARIABLE FIELD COMMENT WITH THE INFORMATION IN THE
    *    VARIABLE LENGTH TRANSACTION FIELD $VAR.
    *
              MVF  $VAR,COMMENT
              HLT
              END
```

EXAMPLE 2

```
$ASP,RPT36Ø,G,8Ø
$RECID,2,7,C1,A
$VAR,31,8Ø,,A
              POOL
    *
    *    THIS LOGIC STATEMENT APPENDS THE INFORMATION IN THE
    *    TRANSACTION FIELD $VAR TO THE INFORMATION IN THE
    *    VARIABLE SET REFER.
    *
```

```
            MVR   $VAR,REFER
            HLT
            END
```

EXAMPLE 3

```
$ASP,RPT360,I,80
$RECID,2,7,C1,A
            POOL
    *
    *    THIS LOGIC STATEMENT WILL SEARCH THROUGH THE VARIABLE
    *    FIELDS (REMARK) OF THE PERIODIC SUBSETS CONTAINING
    *    THE FIELD MEQPT FOR A COMMENT BEGINNING WITH THE
    *    VALUE 01MAY71.  WHEN FOUND, THIS VARIABLE FIELD
    *    WILL THEN BE PRINTED 50 CHARACTERS PER LINE.
    *
                POS   MEQPT,NONE
            NEXT  OVF   REMARK,W1/7,COMP
            COMP  COA   '01MAY71',W1/7
                  BEQ   RESET
                  RVF   REMARK
            STEP  STP   MEQPT,NEXT
                  PRT   'NO MATCH FOUND'
                  HLT
```

FILE MAINTENANCE (FM)

```
NONE    PRT    'NO SUBSETS FOUND'
        HLT
RESET   RVF    REMARK
PUT     OVF    REMARK,W1/50,LAST
        PRT    W1/50
        BRA    PUT
LAST    PRT    W1/50
        HLT
        END
```

7.6 Summary Of POOL Instructions

Data Handling Instructions

Instruction	Operation
MAL A,B	Move Alphabetic
MNU A,B	Move Numeric
MAC A,B	Move Alphabetic Conditional
MNC A,B	Move Numeric Conditional
ADD A,B,C	Addition
ADC A,B,C	Add Conditional
MUL A,B,C	Multiplication
MUC A,B,C	Multiply Conditional
DVD A,B,C	Division
DVC A,B,C	Divide Conditional
SUB A,B,C	Subtraction
SUC A,B,C	Subtract Conditional
CVF A	Clear Variable Data Field
MCT A,B	Move To Record ID
MCS A,B	Move To Secondary Control Field
MCW A,B	Move To Record ID And Write Record
VAL A,B,C	Validity Check
TBL A,B,C,D	Table Lookup Routine
GEN A,B,C,D	Subroutine Processing
COA A,B	Compare Alphabetic
CON A,B	Compare Numeric
SDT A	Store Date-Time Group
MVF A,B	Move To Variable Set/Field
MVR A,B	Move To Variable Set/Field (Input Length).
OVF A B C	Output Variable Field
RVF A	Reset Variable Field Pointer
CLR A	Clear Field

Control Instructions

Instruction	Operation
BRA A	Unconditional Branch
LNK A	Branch And Link
RET	Return Branch
BLT A	Branch If Less Than
BEQ A	Branch If Equal
BLE A	Branch If Less Than Or Equal
BGT A	Branch If Greater Than

FILE MAINTENANCE (FM)

BNE A	Branch If Not Equal
BGE A	Branch If Greater Than Or Equal
HLT	Halt Statement Execution
NOP	No Operation
XNP	Exit No Prior Processing
SMR A	Branch On Summary
BNR A	Branch If New Record
BS2 A	Branch If Switch 2 Is On
BRO A	Branch If Overflow
BNV A	Branch If Invalid
BPO A	Branch If Program Switch On
SOF	Set Overflow Switch Off
SVF	Set Validity Switch Off
SPF	Set Program Switch Off
SRF	Set New Record Switch Off
S2F	Set Switch 2 Off
SOO	Set Overflow Switch On
SVO	Set Validity Switch On
SPO	Set Program Switch On
S2O	Set Switch 2 On
SRO	Set New Record Switch On
SIL	Set Indicator Low
SIE	Set Indicator Equal
SIH	Set Indicator High
NCT	NOP Instruction Counter
END	End Update Statement

Environment Handling Instructions

Instruction	Operation
POS A,B	Activate First Subset
STP A,B	Step Activity To Next Subset
POV A,B,C	Position On Value
STV A,B,C	Step To Value
DSB A,B	Delete Subset And Branch
DSC A	Delete Subset And Continue
BSS A	Built Subset
SSS A	Sequence Subsets
DDR	Delete Data Record

Display Instructions

Instruction	Operation
LOG A	Print On History File
PRT A	Print On History File
PT2 A	Print On History File
WRT A	Write On Auxiliary File
WT2 A	Write On 2nd Auxiliary File
WT3 A	Write On 3rd Auxiliary File
WT4 A	Write On 4th Auxiliary File
WT5 A	Write On 5th Auxiliary File
PCH A	Punch Contents Of A Into Punch Pocket 1
PC2 A	Punch Contents Of A Into Punch Pocket 2

Instructions Used With OM and SODA

Instruction	Operation
BFV A,B	Branch On Transaction Field Valid
BFN A,B	Branch On Transaction Field Not Valid
BTV A	Branch On Transaction Valid
BTN A	Branch Transaction Not Valid
ERR A,B	Log Erroneous Transaction Data

Section 8

ORDINARY MAINTENANCE (OM) EXAMPLES

8.1 Use of Ordinary Maintenance TD Cards

The following logic statement, written entirely in the Ordinary Maintenance language, illustrates a simple method of editing for nonnumeric characters when the type of character in each column is known. (The utilization of additional PICTURES for values to be edited gives greater flexibility, as needed.) The editing function performed is similar to that performed by the POOL logic statement in section 7.5.6. It uses the PICTURE parameter to test for numeric data in the transaction fields, and if the data is correct moves it to the data file. Nonnumeric data is logged on the Ordinary Maintenance error log.

The logic statement also checks the first byte of the transaction record ID field for a legal service code, using the VALUE parameter. If the first byte is in error, the transaction is deleted.

```
$ASP,RPT36Ø,F,8Ø
FIELD RECID 2 7 A CONTROL 1 VALUE J***** N*****
                  M***** N***** ERROR T
FIELD PERS 1Ø 15 D PICTURE NNNNNN ERROR D GEN
FIELD READAVG 2Ø 22 D PICTURE NNN ERROR D GEN
FIELD RITNM 25 27 D PICTURE NNN ERROR D GEN
```

8.2 Use of Ordinary Maintenance TD Cards and POOL Instructions

The following logic statement performs the same function as the one in section 8.1, except that automatic logging on the Ordinary Maintenance log is suppressed. Invalid record control fields are printed on the OM error log by use of the ERR instruction. Logging of other errors is performed by the POOL statements on the normal printer auxiliary output. The BTV instruction is used to exit from the POOL logic when

no errors were found, and the BFN instructions are used to test for invalid fields.

```
    $ASP,RPT36Ø,E,8Ø
    FIELD RECID 2 7 A CONTROL 1 VALUE J***** W******
                     M****** N***** ERROR DS
    FIELD PERS 1Ø 15 D PICTURE NNNNNN ERROR DS GEN
    FIELD READAVG 2Ø 22 D PICTURE NNN ERROR DS GEN
    FIELD RITNM 25 27 D PICTURE NNN ERROR DS GEN

          POOL
          BTV       EXIT
          BFN       $RECID,IDERR
          PRT       T1/8Ø
          BFN       $PERS,ERR1
CHK2      BFN       $READAVG,ERR2
CHK3      BFN       $RITNM,ERR3
EXIT      HLT
ERR1      PRT       'THE PERSONNEL FIELD CONTAINS NON-NUMERIC DATA'
          BRA       CHK2
ERR2      PRT       'THE READAVG FIELD CONTAINS NON-NUMERIC DATA'
          BRA       CHK3
ERR3      PRT       'THE RITNM FIELD CONTAINS NON-NUMERIC DATA'
          BRA       EXIT
IDERR     ERR       $RECID,'INVALID SERVICE CODE'
          DDR
          END
```

Section 9

NEW FILE MAINTENANCE LANGUAGE (NFL)

NFL is a high-level or procedural FM language which provides the user with a language which is easy to learn and simple to use, yet which is powerful and flexible enough to efficiently accomplish a wide range of FM functions.

The NFL section of the FM component accepts, as input, statements written in an English-like language describing the conditions and actions which are to be applied against a NIPS data file for a specified set of update transactions. The language is interpreted and processed (compiled), resulting in an executable logic statement. Error conditions are detected, and diagnostics which will aid the user in correcting the logic statement are printed.

Before reading this section, review sections 2 through 6 of this manual.

9.1 NFL Statement Composition

NFL statements are free-format. Words are separated by blanks, commas, or periods. Multiple statements can be punched on a single card or a statement may be spread over more than one card. Card columns 72-80 must not be utilized. Words, including literals, may not be split over two cards. Statements are composed of statement identifiers, keywords, noise words, labels, and operands. Periods may be used freely for readability. They have no effect on the language processor in this component.

9.1.1 Statement Identifiers

There are a limited number of statement identifiers in NFL which identify a condition, an action, or a point of control. Though the number of statements is limited, a number of functions are implied simply by the structure of the statements. The following lists of identifiers have been grouped by category.

Action Identifiers	Condition/Logic Identifiers	Control Point Identifiers
MOVE	IF	ELSE
ATTACH	AND	CONTINUE
COMPUTE	OR	PROCEDURE
BUILD		END
POSITION		NOTE
LOCATE		NFL
STEP		NFL
PRINT		
PUNCH		
WRITE		
DISPLAY		
DELETE		
DEFINE		
TURN		
GO		
RETURN		

Example -

IF condition AND condition MOVE from location to location, PRINT data CONTINUE

In the preceding example, IF, AND, MOVE, PRINT, and CONTINUE are statement identifiers. The IF identifies a new condition to follow. The AND identifies a condition to follow and denotes that it is a continuation of a "string" of conditions (see section 9.3.1). The MOVE and PRINT identifiers identify actions to be performed if the preceding conditions are true. CONTINUE identifies the point to continue processing regardless of whether the preceding conditions were met.

9.1.2 Keywords

Keywords differ from statement identifiers in that they are embedded in the statement and identify either a secondary action, a specific action from a group of possible actions, or succeeding operands. The following list of keywords are used in NFL. Examples can be found in the section discussing statement descriptions.

Keyword	Usage
NOT	Used to negate a condition
BT BETWEEN	Identifies a between condition
EQ EQUAL EQUALS	Identifies an equal condition
NE	Identifies a not equal condition
LT LESS	Identifies a less than condition
GT GREATER	Identifies a greater than condition
LE	Identifies a less than or equal condition
TAB TABLE	Identifies a validation table condition or conversion of data
PIC PICTURE	Identifies a picture condition
SUB SUBROUTINE	Identifies a data conversion operation
BIT	Identifies a bit condition
ON	Identifies an on (true) condition
OFF	Identifies an off (false) condition

FILE MAINTENANCE (FM)

NEW RECORD	Identifies a new record condition
JOB COMPLETE JOB COMPLETED	Identifies a run complete condition
OVERFLOW	Identifies an overflow condition

Keyword	Usage
+	Add operation
-	Subtract operation
/	Divide operation
*	Multiply operation
=	Denotes equivalency in compute statement
ON (following	PRINT, PUNCH, WRITE, DISPLAY) signifies that a device is being specified
EXIT	Identifies the next operand as an exit label
RECORD	Identifies the object of the action as a record
FIELD	Identifies the object of the action as field
SET	Identifies the object of the action as a set
SUBSET	Identifies the object of the action as a subset
OVER PAST BEYOND NEXT FIRST	Specifies resulting position of set.

9.1.3 Noise Words

Noise words are a group of commonly used words which have no effect on the functions of the statement but which help to give the statements a more English-like readability. Noise words can appear any place within a statement. The following noise words are allowed in NFL:

A	FOR	IS	WITH
AN	FROM	THAN	USING
AS	IN	THE	
BY	INTO	TO	

9.1.4 Statement Labels

The provision for labeling statements (or procedures) allows a name to be associated with a statement. These names can then be referenced as exit points or to change the sequence of execution. Labels must begin with an alpha character and may contain only alphameric characters. The label can contain up to seven characters. It is identified as a label by suffixing the name with a colon. A valid label must have a space following the colon. Statement identifiers or noise words cannot be used as labels.

Example -

```
LOOP: MOVE....GO TO LOOP
```

LOOP is a label associated with the MOVE statement. In the example, the GO causes the order of execution to be changed to the statement labeled LOOP.

The following statements may not have labels:

An If statement or any part of an IF statement such as the AND, OR, OR ELSE clause.

A CONTINUE statement

A NOTE statement

A DEFINE statement

9.1.5 Operands

Statement operands identify control locations, subroutines and tables, literal values, and data locations.

9.1.5.1 Control Location Operands

Control locations are references to statement labels (section 9.2.4). This type of operand is an exit point or a change (GO) in execution sequence.

Example 1 -

```
LOCATE SET MEQPT,EXIT TO NOSUB
```

In this example, if there are no subsets belonging to the set MEQPT, the next statement to be executed would be the statement with the label NOSUB. Thus NOSUB is a "control location" type operand.

Example 2 -

```
GO TO XYZ
```

When the above statement is executed, the next statement to be executed would have the label XYZ. XYZ is a "control location" type operand.

9.1.5.2 Subroutine/Table Name Operands

When user written subroutines or tables are to be executed, the subroutine or table name is designated as a statement operand.

Example -

```
IF LOC IS IN TABLE PLACES
```

In the above example, the operand PLACES names a user written validation table.

9.1.5.3 Literal Value Operands

When a value is specified, it is a literal value operand. Alpha or numeric literals can be specified. To designate an alpha literal, the value must be enclosed in quotes. Though several words may be included within the quotes, it is considered as a single operand (care must be taken not to split a literal value operand over more than one card). With the exception of the quote and ampersand characters, any character can be used within the quotes. To define a literal value operand containing a quote or ampersand, a double quote or ampersand must be used. In this instance, the redundant special character is not counted in determining the length of the literal.

Numeric literal value operands are designated by expressing a numeric value (not enclosed in quotes). A + (plus) or - (minus) sign may be prefixed to the value. If the sign is missing, the value is assumed to be positive.

Example 1 -

```
MOVE 'ROSSLYN PLAZA' TO LOC
```

In the above example, the alpha literal value ROSSLYN PLAZA will be moved.

Example 2 -

```
COMPUTE SCALE = +1000 * LENGTH
```

In the above example, the numeric literal 1000 will be used in the computation. Note that in the example the plus sign could be omitted.

9.1.5.4 Data Location Operands

Data location operands are designated by using symbolic names which have been defined to the system in such a way that the name is equated to the location and length of the data.

The location and length may be modified or adjusted for these types of operands (except for binary, coordinate and

variable field data) by use of partial field notation. This is done by following the symbolic name (separated by at least one blank or comma) with the desired beginning and ending character positions. The form of expression for partial field notation is N/M where N is the beginning character position and M is the ending position; e.g., to designate the first two characters of the field MEQPT, specify MEQPT 1/2.

There are four basic types of "Data Location" operands in NFL. These are file data, transaction data, indirect data, and defined constants and areas.

9.1.5.4.1 File Data Operands

File data operands are designated by the symbolic name assigned to the specific value (field) during file structure.

Example -

IF MEPSD IS EQUAL TO MERDY

MEPSD and MERDY are file data operands.

9.1.5.4.2 Transaction Data Operands

Transaction data operands are designated by the symbolic name assigned to the value (field) in the statement transaction descriptor deck. The symbolic name is always prefixed with a $ character. Note: NFL does not permit the TN/M form of transaction references.

Example -

MOVE $MEQPT TO MEQPT

$MEQPT is a transaction data operand.

9.1.5.4.3 Indirect Data Operands

Indirect data operands (see section 7.2) are designated by prefixing the symbolic name (defined by the transaction descriptor deck) with C$. Partial field notation is not allowed for indirect data operands.

Example -

```
POSITION  TO THE FIRST SUBSET FOR C$ZILCH,EXIT TO NOS...
```

C$ZILCH is an indirect data reference.

9.1.5.4.4 Defined Constant and Area Operands

Constants and areas which can be used as work areas can be defined and given symbolic names by the user. These are then referenced by symbolic name as operands in statements.

NFL provides three constants which can be referenced and need not be defined by the user. These have the following characteristics and symbolic names.

a. For a value of zero, use ZERO, ZEROS, or ZEROES.

b. For a value of blank, use BLANK or BLANKS.

c. For the current date and time, use SYSDATE.

SYSDATE is in the form YYDDDTTTT where Y is year, D is Julian date and T is time. Partial-field notation will not be allowed for the zero and blank values.

9.2 Special Requirements and Considerations

There are several special requirements or considerations which are imposed by NFL. These must be clearly understood by the user for effective application of NFL.

9.2.1 Data Mode Compatibility

The NFL statements are not "mode" oriented, therefore it does not require one statement for alpha data and another for numeric. Obviously, only numeric data may be used in an arithmetic statement. In all other instances, the system checks for the mode of the data and determines the mode of the operation. EBCDIC work areas are nonmode associated and take on the mode of the related data. If, when two operands are specified, both are nonmode associated, the mode defaults to alpha. There are some combinations of mode which are illegal. Legal mode combinations are given in the table below.

Mode codes are:

A - alpha

B - binary

C - coordinate

D - decimal

W - nonmode associated

Legal Mode Combination:

Operand 1	Operand 2	
A	A	
A	C	
A	W	
B	B	
B	D	
B	W	
C	A	
C	C	
C	W	
D	A	Note the combinations
D	C	cannot be reversed
D	B	
D	W	
D	D	
W	W	
W	B	
W	C	
W	D	
W	A	

9.2.2 Data Length Compatibility

When two operands for a statement require data length compatibility, NFL automatically pads or truncates to obtain this compatibility. The second operand length is the determining length. Alpha data fields are padded with blanks or truncated on the right. Numeric data fields are padded with zeros or truncated on the left.

9.2.3 Special Statement Sequence Requirements

There are several NFL statements or groups of statements which require special sequence considerations. These are described in the paragraphs which follow.

9.2.3.1 Condition/Action Statement Sequence

Condition statements are composed of conditional clauses logically connected and identified by IFs, ANDs, and ORs. An action(s) is always associated with a condition. This action is executed if the condition is true. There may also be a set of "false" actions associated with a condition. False actions are optional and identified by an ELSE statement. After the "true" or "false" actions have been executed, the point at which execution is continued is identified by a CONTINUE statement.

There may be multiple "true" and "false" statements. All statements between the last condition clause and the ELSE or CONTINUE statements are "true" actions. All statements between the ELSE and the CONTINUE are "false" actions.

A second condition cannot appear between the first condition and the actions associated with it.

For labeling purposes, a condition/action statement sequence is considered to be a single statement. A label may precede the keyword IF but no further labels are permitted until after the keyword CONTINUE.

Example 1 -

```
IF MEPSD IS NOT GT MEREQ,AND MEQPT IS EQUAL TO
'PLANE', PRINT '***',MEQPT,'***DEFICIENCY***',
MEPSD,MEREQ.
CONTINUE,POSITION....
```

In the preceding example, if the condition is true the action PRINT will be executed. If the condition is false, or on completion of the PRINT action, processing will proceed with the CONTINUE statement.

Example 2 -

```
IF MEPSD IS NOT GT MEREQ,AND MEQPT IS
EQUAL TO 'PLANE', PRINT '***',MEQPT,
'***DEFICIENCY***',MEPSD, MEREQ.
ELSE, COMPUTE DIF = MEPSD - MEREQ.
CONTINUE, POSITION...
```

In the preceding example, if the condition is true, the action PRINT will be executed. If the condition is false, the action COMPUTE will be executed. In either case, execution will then continue at the CONTINUE statement.

9.2.3.2 Procedure Definitions

Procedures are simply a method of grouping conditions and actions as a unit and allowing these to be executed as a unit. The first statement of a procedure is the PROCEDURE statement. The last statement is an END statement. All condition and action statements between the PROCEDURE and END statements are considered as part of that procedure.

A procedure can be executed by executing a GO to the procedure from outside of the procedure or by "dropping" in it. Upon completion, a procedure will return control to the statement following the GO statement (GO type execution) or the statement following the procedure END statement ("dropping" type execution). Exit (return) from a procedure occurs when the RETURN statement is executed or when the end of the procedure (END statement) is encountered.

Procedures are restricted in the following sense: execution of a procedure can only be initiated at the entry point of the procedure. Nesting (procedure calling a procedure) is not permitted.

The examples below illustrate the two methods for executing a procedure.

Example 1 -

```
... GO TO SUM, MOVE RESULT TO TOTAL...
...SUM: PROCEDURE conditions and actions END...
```

The preceding example illustrates how a procedure is executed from a GO statement.

The procedure named SUM would be executed when the GO TO SUM statement is executed. When the END statement within the procedure is executed control will be returned to the next statement following the GO TO SUM statement. Following

the execution of the procedure, the next statement to be executed would be the MOVE RESULT TO TOTAL statement.

Example 2 -

```
...MOVE SYSDATE TO DATE, CONVERT: PROCEDURE...
procedure conditions and actions RETURN additional
procedures conditions and actions END PRINT 'FILE
UPDATED', DATE...
```

The preceding example illustrates the "drop through" method of executing a procedure. The procedure is executed following the statement MOVE SYSDATE TO DATE. When the RETURN statement is executed, the next statement to be executed would be the first statement following the END procedure statement which in the example is a PRINT statement.

9.2.3.3 Define Sequence Requirements

There are two simple considerations regarding sequence requirements for use of DEFINE. The first is that a constant or area must be defined before it can be referenced by another statement. The second involves the define and initialize (VALUE) statement. The initialization occurs at the point that the statement is encountered. Thus, in a logic statement which loops back, an area might be reinitialized or, if the path of execution never encountered the initialize statement the area would not be initialized during that execution.

Though not required, these two sequence requirements can always be satisifed by placing all define statements at the beginning of the NFL logic statement.

9.2.4 Subset Positioning

Special consideration should be given to the subset positioning actions which position to the next subset. If, at the time the statement is executed, the set is in an inactive status (either the set has never been activated or the set has been positioned past the last subset) the first

subset of the set will be activated. If there are no subsets, the exit will be taken.

9.3 NFL Statement Description

This section describes each of the NFL statements in detail. In each of the examples which are given, required terms are underscored. Those which are not required, such as noise words, are not underscored. Commas and periods are optional; i.e., they are not required and are effectively ignored. A summary of the syntax for the NFL can be found in section 9.4.

9.3.1 Conditional Statements

To review what has previously been stated regarding NFL conditional statements:

a. The statement identifier IF introduces the first condition clause of a new conditional "string".

b. AND or OR identifiers identify a continuation of the conditional string and the logic to be applied between clauses.

c. A conditional string is always followed by a "true" action(s), optionally a false action(s), and a required continuation point.

d. A second condition may not appear before the continuation point.

Several types of conditional clauses in NFL are listed below.

Relational

Table Validation

Picture Mask

Switch

Bit Mask

New Record

Run Complete

Overflow.

9.3.1.1 Relational Condition

The relational condition is used to compare two pieces of data for a stated relationship. The clause is composed of an operand, a relational operator, and a second operand.

Example -

IF FIELDA IS EQUAL TO FIELDB...

The above example is a typical example of the relational condition clause. The operand represented as FIELDA can reference transaction data, data file data, indirect data, constants, or work areas. The relational operator (EQUAL) could be any one of the following:

EQ EQUAL EQUALS	The relationship must be equal to be true.
NE	The relationship must be unequal to be true.
LT LESS	The relationship must be less to be true.
GT GREATER	The relationship must be greater to be true.
LE	The relationship must be equal to or less to be true.
GE	The relationship must be equal to or greater to be true.

FIELDB can reference transaction data, data file data, indirect data, constants, work areas or literal values.

FIELDB might also be expressed as a multiple value operand for the equal (and not equal) relationship.

Example -

...AND FIELDA IS EQUAL TO 'ROSSLYN', 'GBURG', 'WASHINGTON',...

The multivalue equal condition is processed as an OR string, i.e., if any one of the multiple values satisfies the desired relationship, the clause is true. This, of course, is just the opposite if the negative relationship is required.

Example -

...AND FIELDA IS NOT EQUAL TO 'ROSSLYN', 'GBURG', 'WASHINGTON',...

In this example, for FIELDA to be true, it must not equal 'ROSSLYN', 'GBURG', or 'WASHINGTON'.

The between relationship condition requires two fields or values following the specified relation.

Example -

...OR FIELD IS BETWEEN 500/700...

The two between values are separated by a slash; multiple value operands are allowed with the between relationship.

Example -

IF FIELD IS BETWEEN 500/700, 900/1100...

The results (true or false) of the multiple value between is the same as described for the equal relationship. The between relation operator can be expressed as BT or BETWEEN.

When specifying partial field notation with the second operand of the between, care must be taken to be sure that

the partial field notation is preceded and followed by at least one blank (see section 9.5).

9.3.1.2 Table Validation

The table validation condition allows the user to use a user-written subroutine or table lookup for validation purposes. The clause is specified by designating the data to be validated and the table (or subroutine) which is to be executed to perform the validation.

Example -

...AND FIELDA IS IN TABLE CNTRYS...

FIELDA may be transaction data, data file data, indirect data, constants or work areas. The keyword TABLE could also be designated as TAB. CNTRYS is the name of the table or subroutine.

NOT could be used to negate the condition.

Example -

...OR FIELDA IS NOT IN TABLE CNTRYS...

The validation must be unsuccessful for the above clause to be true.

9.3.1.3 Picture Mask

The picture condition allows an alpha or decimal value to be tested for designated character types. A picture mask must be specified. This mask is composed of a series of the following characters.

Characters	Test
A	Alpha characters
N	Numeric characters
S	Special characters

B	Blank characters
X	Nonblank characters
Y	Nonspecial characters
*	No check.

The field to be tested is checked character-by-character for the condition specified by the corresponding character in the mask.

Example -

...AND FIELD IS AS IN PICTURE 'NNAAANN'

The above example would test the contents of FIELD for two numeric followed by three alpha followed by two numeric characters.

FIELD may reference transaction data, data file data, indirect data, constants or work areas. The mask must be an alpha literal value.

9.3.1.4 Switch Test

The switch test is used with the TURN action (see section 9.3.2.8). It is a tool for testing for an ON/OFF condition of a switch which is set by the user.

The switch is a single byte (character) in core which is set to an EBCDIC zero for OFF and an EBCDIC one for ON. The switch is designated by a symbol which can be defined with a DEFINE statement by the user or can be automatically defined by the system.

The form of the switch clause is operand, followed by the keyword OFF or ON.

Example -

...OR THE GOSWTCH IS ON...

In the preceding example, if the character represented by the symbol GOSWTCH contains a one, the clause will be true.

The switch operand can only be a defined area (by the user or system).

The keyword (ON) can be ON or OFF.

9.3.1.5 Bit Mask Test

The bit mask test allows a user to scan a field on a character basis, while testing for the presence of designated bits within a character. The bit mask is specified as a series of ones and zeros. If less than eight are specified, zeros will be padded to the right. If more than eight are specified, they will be truncated on the left.

When an ON (or NOT OFF) condition is specified, if any bit in any character of the field being scanned matches, the result is true. For an OFF (or NOT ON) conditon, if all of the designated bits in any character of the field being scanned are off, the result is true.

The form of the bit test is; operand, keyword (BIT), bit mask, keyword (ON, OFF, NOT ON, or NOT OFF).

Example -

IF FIELD BIT 10111111 IS ON...

In the preceding example, if any bit of any character of FIELD is a one, other than the second bit, the clause will be true.

Example -

IF FIELD BIT 10111111 IS OFF...

In this example, a true condition will result only when all of the bits, or all bits but the second, are zero for any character of FIELD.

The bit mask is designated as an unsigned numeric literal value composed of ones and zeros.

9.3.1.6 New Record Test

When a file record cannot be found for a transaction, a new file record is automatically created by the File Maintenance capability. There are times when the user would like to know when this condition exists. This can be determined by the new record test. The example below illustrates how to specify this condition.

Example -

...AND NEW RECORD...

The next example illustrates how a test for an old record could be designated by use of the condition negation.

Example -

...OR NOT NEW RECORD...

9.3.1.7 Job Complete Test

For the user who wishes to perform an action at the end of the maintenance run, the job complete test is provided. This test is particularly useful in producing "maintenance summaries". It would only be used in Range logic statements.

The following example illustrates how the job complete test would be used to test for the end of the run and for not the end of the run.

Example -

...IF THE JOB IS COMPLETE... ...AND THE JOB IS NOT COMPLETE...

9.3.1.8 Overflow Test

The test for OVERFLOW statement is used after a compute statement to determine whether overflow has occurred. The test must be made before a second COMPUTE statement is executed or the status of the first statement is lost. The example following illustrates the overflow test.

Example -

...IF OVERFLOW IS ON... ...IF OVERFLOW IS OFF...

9.3.2 Action Statements

Action statements specify a function to be performed. The operations which they specify to be performed may be unconditional or they may be based on the satisfaction of a prior condition.

9.3.2.1 Data Movement

There are two statements which can be used to move data from a specified source to a specified destination.

9.3.2.1.1 The MOVE Statement

There are two forms of the MOVE statement. The first is a simple movement of the data from a source location to a destination location. The second is the movement of the data from a source location via a conversion subroutine or table.

Example -

...MOVE FIELDA TO FIELDB...

The preceding example illustrates a simple MOVE statement. FIELDA may reference transaction data, data file data, indirect data, defined constants, defined work areas or literal values. FIELDB may reference data file data, indirect data or defined work areas. Data mode compatibilities will automatically be checked and field lengths will automatically be adjusted. If partial field notation is designated, it will be checked to determine whether it is within the boundaries of the data field. When moving coordinate fields, to or from EBCDIC fields, conversion to and from internal format automatically occurs.

Example -

...MOVE FIELDA TO FIELDB USING TABLE STATES,
EXIT ILLST...

The preceding example illustrates the movement of data through a conversion table. FIELDA and FIELDB may reference the same types of data as for the simple move. The data characteristics for FIELDA are checked against the input characteristics for table STATES and the FIELDB data characteristics are checked against the output characteristics for table STATES. The exit label (ILLST) is the label of the NFL statement to which control will be given if the table is not successful in converting the data.

In either of the MOVEs, FIELDA may not be a variable data file field or set. If FIELDB is a variable data file field or set, any existing variable data in FIELDB will be replaced by the contents of FIELDA. If FIELDB is a major control field, a warning diagnostic will be printed when the logic statement is compiled and the move will be allowed. IF FIELDB is a secondary control field (subset ID), the move will be allowed without any warning diagnostic to the user.

Note: When alpha or decimal data is moved to an alpha or decimal field, the 'MOVE' instruction uses a 360 ALC MOVE instruction which moves left to right through each field one byte at a time. Therefore, caution must be used whenever overlapping portions of the same field are used as the operands of a 'MOVE' instruction (e.g. MOVE FIELDA 2/4 TO FIELDA 3/5...).

9.3.2.1.2 The ATTACH Statement

The ATTACH statement is used only to move data to a variable set. It differs from the MOVE to variable set in that instead of replacing the existing contents, the data to be moved is appended to the existing data.

Example -

...ATTACH FIELDA TO FIELDB...

In the preceding example, assuming that FIELDB is a variable set, the contents of FIELDA would be appended to the existing contents of FIELDB. FIELDA may reference transaction data, data file data, indirect data, defined constants, defined work areas or literal values. FIELDB may reference data file data which are variable sets.

9.3.2.2 The COMPUTE Statement

The COMPUTE statement is used to specify one or more arithmetic operations and to store the result in the designated result field. The general format of the COMPUTE statement is the result field followed by the = (equal character) followed by the arithmetic expression.

Example -

COMPUTE FIELDA = Arithmetic Expression

In the preceding example, the final result of the expression to the right of the = character will be placed in the location designated as FIELDA. FIELDA may reference data file data, indirect data or defined work areas.

The arithmetic expression can consist of operands separated by arithmetic operators. The operands may reference transaction data, data file data, indirect data, defined work areas, defined constants or numeric literals. The arithmetic operator may be a + - * / character indicating addition, subtraction, multiplication, or division. The arithmetic operator must be preceded and followed by a blank character.

Parentheses may be used to alter the sequence of arithmetic operations. Expressions within parentheses are evaluated first. When parenthesized expressions are in a nest of parentheses, evaluation begins at the innermost level and continues until the outermost parenthesis level is reached.

The multiplication and division operators are at a higher precedence level than the addition and subtraction operators. In expressions containing consecutive equal precedence operators, evaluation will be performed from left to right.

Note: Only integer arithmetic may be performed. Division of a value by a larger value produces a zero result. Therefore, careful consideration must be given to the sequence of operations. Full word binary logic is used, so the maximum value resulting from any operation is restricted to ±2,147,483,647.

Example -

```
A+B-C*D/E
```

The preceding expression, because of the order of processing would have the same result as

```
(A+B)-((C*D)/E)
```

while

```
A+B*C-D/E
```

would have the same result as

```
(A+(B*C))-(D/E)
```

9.3.2.3 Subset Positioning Statements

To reference a data field belonging to a periodic set, that set must be activated or be pointing to a subset belonging to that set. NFL provides three statements to perform the functions of activating and "stepping" through a set; each statement has an exit. The exit designates the

label of the statement to be given control if there are no subsets or if a set becomes exhausted.

9.3.2.3.1 The LOCATE Statement

The LOCATE statement will activate the first subset of a set.

Example -

...LOCATE SET FIELDA , EXIT NOSS...

The preceding example would cause the first subset of the set in which FIELDA belongs to be activated. If there are no existing subsets for that set, control would be given to the statement following the label NOSS. FIELDA may reference a data file field, indirect data, or the actual set number may be designated. The exit label, NOSS, may be the label of any statement other than a procedure label.

9.3.2.3.2 The STEP Statement

The STEP statement will cause the next subset of a set to be activated.

Example -

...STEP SET FIELDA , EXIT ENDSET...

In the preceding example, the next subset of the set in which FIELDA is a field will be made active. If there are no other subsets to be made active the exit will be taken. FIELDA may be referenced as a data file field, indirect data or as the actual set number. The exit label, ENDSET, may be the label of any statement other than a procedure label.

9.3.2.3.3 The POSITION Statement

The POSITION statement can be used for all subset positioning. The basic positioning actions which can be done with the POSITION statement are:

a. The POSITION statement can be used to position set to the first subset within the set.

Example -

...POSITION TO THE FIRST SUBSET FOR FIELDA , EXIT NOSS...

The preceding example would cause the first set of the set in which FIELDA belongs to be activated. If there are no subsets belonging to that set, control will be given to the statement following the label NOSS which must not be a procedure label. FIELDA may reference a data file field, indirect data or as the actual set number.

b. The POSITION statement can be used to position a set to the next subset in a set.

Example -

...POSITION TO THE NEXT SUBSET FOR FIELDA , EXIT TO ENDSET...

The preceding example would cause the next subset for the set in which FIELDA belongs to be made active. If there is no additional subset, control will be given to the statement following the label ENDSET (must not be a procedure label). FIELDA may reference a data file field, indirect data or be the actual set number.

c. The POSITION statement can be used to position a set after the last subset in a set. This is a useful tool to build a new subset at the end of a set.

Example -

...POSITION AFTER LAST SUBSET FOR FIELDA...

The preceding example would cause, in effect, the next action for that set to be after the last subset of the set (the only valid action against that set would be build subset or delete set). Note that there is no exit in the example. An exit can be specified; however, the exit will never be executed, i.e., it is ignored. FIELDA may

reference data file data, indirect data, or be the actual set number.

d. The POSITION statement can be used to position a set to the subset in which a designated field of that set contains a designated value.

Example -

...POSITION TO FIELDA IN FIELDB, EXIT TO NOHIT...

The preceding example would cause the following:

If the set in which FIELDB belonged was inactive, the first subset would be activated and the contents of FIELDB compared with the contents of FIELDA. If an equal condition exists that subset will be activated. Otherwise, the set containing FIELDB will be stepped and the compare made until an equal condition exists or the set becomes exhausted.

If the set has no subsets or if it is exhausted without an equal condition, control will be given to the next statement following the NOHIT label (may not be the label of a procedure).

If when the statement is executed, a subset of the set to which FIELDB belongs is active, the subset will be stepped and the compares will commence with that subset.

Note: If a set is stepped past the last subset (effectively the set becomes inactive), the next execution of the POSITION statement will cause the search to begin with the first subset.

In the preceding example, FIELDA may reference transaction data, data file data, indirect data, defined constants, defined work areas or literal values. FIELDB may reference a data file field or indirect data.

9.3.2.4 Auxiliary Output Statements

There are four forms of auxiliary output available in NFL. The statements to format and output each form are identical except for the statement identifier and the number of devices which can be designated. The data to be output is specified as a list. NFL automatically formats the data as a continuous string of data. If the value is binary, it will be converted to an EBCDIC value. The converted length of a binary data file value will be the length designated when the file was structured. The converted length of a binary work area will be 10 characters. Coordinates which are in internal form will be converted to external form. If the coordinate was defined as a field, the resulting length will be that designated when the file was structured. If the coordinate was defined as a group, the resulting length will be 15 characters per point. Blanks are not automatically inserted between data values. If they are desired, they must be designated by the user. If the system provided constants, BLANK, BLANKS, ZERO, ZEROS or ZEROES are designated for output, the length will be one character. The list of operands to be output may be transaction data, data file data, defined constants, defined work areas or literal values. The total number of bytes specified by all auxiliary output instruction operands cannot exceed 994. NIPS adds six bytes to the specified data and the DCB LRECL is 1000. Violation of this limit will result in a System 001 ABEND (I/O ERROR, RECORD TOO LONG).

9.3.2.4.1 The PRINT Statement

The PRINT statement is used to log data on a printer during a FM update run. Two printers can be designated. If a printer is not designated in the PRINT statement, the default is printer 1. In the TP mode, printer 1 designates the display terminal and printer 2 an OMQ display device.

Example -

```
...PRINT FIELDA,'   ', FIELDB,'   ',FIELDC...
...PRINT ON 2, FIELDA,FIELDB,FIELDC.
```

The first example illustrates the PRINT statement defaulting to printer 1; the second illustrates the method for specifying the printer.

Note: The printer specification (on N, where N is 1 or 2) must immediately follow the statement identifier. If the total length of data is greater than 132 characters, lines of 132 characters will be printed until the total length is exhausted.

9.3.2.4.2 The PUNCH Statement

The PUNCH statement is used to punch data onto cards. Two punches can be specified. If a punch is not specified, default is to punch 1.

Example -

...PUNCH FIELDA, FIELDB, FIELDC...

The preceding example would result in the contents of FIELDA, FIELDB and FIELDC being punched in cards. If the total length is greater than 80 characters, 80-character records will be punched until the total number of characters is exhausted.

9.3.2.4.3 The WRITE Statement

The WRITE statement is used to output data on a sequential output file. As many as five output devices can be designated. If the device is not designated, it defaults to auxiliary device 1; i.e., the device specified on the FM.AUX1 DD card.

Example -

WRITE ON 3, FIELDA, FIELDC...

The preceding example would cause the contents of FIELDA, FIELDB and FIELDC to be formatted and a record equal to the total length output on auxiliary output device 3.

9.3.2.4.4 The DISPLAY Statement

The DISPLAY statement is provided to assist the terminal operator in correcting erroneous transaction data when using the online update capability of Source Data Automation (SODA).

Example -

....DISPLAY $FIELD, 'TUBE MESSAGE'..

When this instruction is executed during a SODA run, it causes the displayed transaction field ($FIELD) to be underscored with a key to the message provided as the literal 'TUBE MESSAGE'.

When this instruction is executed during a batch FM run, it causes the message to be printed on the Ordinary Maintenance error log (see section 6.2.9).

9.3.2.5 The BUILD Statement

The BUILD statement causes a new subset to be created. The new subset is created at the point where that set is active; i.e., if the set has not been positioned after the last subset processed, existing subsets are "pushed down" in the set and the newly created subset inserted at the active point. After the subset has been created, the new subset is made active. No data is moved to the subset as a result of the BUILD statement.

Example -

...BUILD SUBSET FIELDA...

The preceding example causes a new subset to be built for the set in which FIELDA belongs. FIELDA may be data file data, indirect data or the actual set number.

9.3.2.6 The DELETE Statement

The DELETE statement is used either to delete the record, to delete a set, to delete the currently active subset, or to clear a field.

a. The DELETE record is illustrated in the following example.

Example -

...DELETE RECORD...

The preceding example informs FM that the current data record is to be deleted. After execution of this instruction, no further processing is performed against the current data record and a return is made to FM.

b. The DELETE set is illustrated in the following example.

Example -

...DELETE SET FOR FIELDA

The preceding example causes the entire set to which FIELDA belongs to be deleted. FIELDA may be data file data, indirect data or the actual set number.

c. The DELETE subset is illustrated in the following example.

Example -

...DELETE THE SUBSET FOR FIELDA...

The preceding example would cause the currently active subset for the set in which FIELDA belongs to be deleted. After the subset is deleted, the next subset (if any) is made active. FIELDA may be data file data, indirect data or the actual set number.

d. The DELETE field is illustrated in the following example.

Example -

...DELETE FIELD FIELDA...

The preceding example would cause one of the following results depending on FIELDA:

- o If numeric, FIELDA would be set to zero.
- o If alpha, FIELDA would be set to blanks.
- o If a coordinate, FIELDA would be set to zero.
- o If a variable field or set, the variable field or set would be deleted.

9.3.2.7 The DEFINE Statement

The DEFINE statement is used to define constants and work areas. A constant is simply a way to define a literal value which can be referenced with a symbol. Data cannot be moved to a constant. Work areas are just what the name implies, and are used for the temporary storage of data.

There are two types of work areas. One is a system provided logic statement work area consisting of a 999-byte EBCDIC area and a 20 full word binary area. Data can be passed from one logic statement to another or retained between data records by use of this type of work area. The second type, or logic statement internal work area, is unique to the logic statement in which it is defined. Any data which is moved to it is lost between data records.

Each define assigns a symbol to the work area or constant. The symbol is then used to reference the area or constant. An error will occur if the same symbol is defined more than once or if a defined symbol is the same as a data file field name.

9.3.2.7.1 Defining a Constant

The following examples illustrate how a constant is defined.

Example -

...DEFINE HEADER AS 'LIST OF RECORD IDS UPDATED'...

...DEFINE PI AS +314...

In the first example, HEADER is the symbol assigned to the alpha literal value. When that symbol is referenced in a NFL statement the literal value will be used. In the second example, PI is the symbol assigned to the numeric value.

9.3.2.7.2 Defining an Inter-logic Statement Work Area

To define an area in the system provided EBCDIC work area, one identifies the symbolic name to be assigned to the area and the relative positions within that area in the form WN/M where N is the relative position of the first character and M is the relative position of the last character.

Example -

...DEFINE SAVE AS W71/80...

In the preceding example, SAVE is the symbolic name assigned to characters 71 through 80 of the EBCDIC work area. These characters can then be referenced by the NFL condition and action statements by the symbolic name, SAVE.

Defined areas in the EBCDIC work area can be overlapped.

Example -

...DEFINE DAY AS W1/2...

...DEFINE MONTH AS W3/5...

...DEFINE YEAR AS W6/7...

...DEFINE DATE AS W1/7...

In these examples, DATE overlaps DAY, MONTH, and YEAR.

A binary work area is assigned a symbolic name by identifying the name followed by the designated word in the form BN where N is a word number between 1 and 20 inclusive.

Example -

...DEFINE COUNT AS B4...

In the preceding example, COUNT is the symbolic name to be assigned to the fourth binary word in the system-provided binary work area.

9.3.2.7.3 Defining an Intra-logic Statement Work Area

A logic statement internal work area is defined by specifying the symbolic name and designating the number of characters to be assigned to that symbolic name. The number of characters is designated by the form #N where N is the number of characters to be assigned.

Example -

...DEFINE HOLD AS #10/...

In the preceding example, a 10-character area will be reserved and can be referenced using the symbolic name HOLD.

9.3.2.7.4 Defining and Initializing an Area

A work area can be defined and initialized with a value each time that the logic statement is executed. When using this capability, the define sequence requirements (section 9.2.3.3) should be considered. To specify that a defined area is to be initialized with a value, the area definition (sections 9.3.2.7.2 and 9.3.2.7.3) is followed by the keyword VALUE followed by the literal value or one of the system-provided constants (SYSDATE, ZERO, ZEROS, ZEROES, BLANK, or BLANKS).

Example -

...DEFINE SAVE AS W71/80, VALUE IS BLANK...

In the preceding example, a 10-character area in the system work area will be assigned the symbolic name SAVE. That area will be initialized to blanks each time the logic statement is executed.

Example -

..DEFINE COUNT AS B4, VALUE IS 0...

In the preceding example, the binary work area will be assigned the symbolic name COUNT. Each time that the logic statement is executed, it will be initialized to zero.

Example -

..DEFINE HOLD AS #10, VALUE IS 'ABCDEFGHIJ'

In the preceding example, the 10-character work area will be assigned the symbolic name HOLD. Each time that the logic statement is executed, it will be initialized to the value ABCDEFGHIJ.

9.3.2.8 The TURN Statement

The TURN statement is used to set a 1-character area to an ON or OFF status. The area may be defined by the user with a DEFINE statement or he can let the system define it for him. If the system defines the area, it will be a logic statement internal work area.

The TURN statement is primarily for use with the switch test condition. The user designates the switch setting he desires to be set with the TURN statement. Later, he can test for that condition with the switch condition. The switch is set to an EBCDIC zero for off or one for on.

Example -

...TURN SWITCHA ON...

In the preceding example the area (switch) having the symbolic name SWITCHA will be set to an EBCDIC one. If no area has been defined with the symbolic name SWITCHA, an

area will automatically be defined and given the symbolic name SWITCHA.

SWITCHA may only reference defined (by the user or system) areas.

9.3.2.9 Execution Sequence Changing Statements

There are two NFL statements (not counting exits from other action statements) which alter or change the sequence of executing statements.

9.3.2.9.1 The GO Statement

Normally, NFL statements are executed sequentially in the order that they are read into the system. The GO statement can be used to change that order. Only a statement label can be designated as the point to continue execution. That label may be a procedure label (unless the GO is inside a procedure definition (see section 9.2.3.2)) or a statement label. If it is a statement label, it must be at the same level (within or without a procedure) as the GO statement. If a GO to a procedure label is executed, return from the procedure will be to the next sequential statement following the GO. If it is not a procedure label, control is not automatically returned.

Example -

```
...GO TO LOG ELSE...          ...LOG: PRINT...
```

The preceding example illustrates a typical use of the GO statement. Assuming that a condition precedes the GO statement, if the condition is true, the order of statement execution is changed to the statement following the label LOG. It is not a procedure statement, thus control will not be returned.

Example -

```
...GO TO SUM, MOVE...         ...SUM: PROCEDURE...
```

The preceding example illustrates the use of the GO statement to execute a procedure. When the procedure has completed execution, execution would resume with the MOVE statement following the GO statement.

9.3.2.9.2 The RETURN Statement

The RETURN statement when used within a procedure returns control to the mainline statement or when used in the mainline, terminates execution of the logic statement. If control by the procedure was gained from a GO statement (see section 9.2.3.2), execution of the RETURN statement would cause control to be returned to the next statement following the GO. If control was gained by the procedure by the "drop through" method, execution of the RETURN statement will cause control to be returned to the next statement following the procedure END statement.

The RETURN statement has no operands. Examples in section 9.2.3.2 illustrate the use of the RETURN statement.

9.3.3 Control Point Identifiers

There are several statements which cause no condition to be satisfied or action to be performed. Their primary function is to identify statement groupings or control points.

9.3.3.1 The NOTE Statement

The NOTE statement actually is not even a control point. It is simply a means for the user to insert commentary text between NFL statements.

The NOTE statement has one operand. It is an implicit literal enclosed in quotes. The NOTE statement can appear between any two statements but should not appear between a label and its associated statement.

Example -

```
...MOVE DATA 3/5 TO MONBUC NOTE 'MONTH ONLY TO WORK
```

AREA'

In the preceding example, the NOTE statement is used to explain the characters being moved.

9.3.3.2 The PROCEDURE Statement

The PROCEDURE statement identifies the beginning of a group of statements which will be treated as a unit. The PROCEDURE must be preceded by a label. This label is called the procedure name.

9.3.3.3 The END Statement

The END statement identifies two control points. It identifies the end of a procedure and/or it identifies the end of the logic statement. It has no operands.

Example -

```
NFL Conditions and Actions

PROC1: PROCEDURE
Conditions and Actions
END

PROC2: PROCEDURE
Condition and Actions
END

END.
```

The preceding example illustrates a logic statement containing two procedures. The first END statement encountered terminates the group of statements which comprise procedure PROC1. The second END statement does the same for the procedure PROC2. The third END statement terminates the entire logic statement.

9.3.3.4 The ELSE Statement

The ELSE statement identifies the beginning of a group of action statements which are to be executed only if a preceding condition was false. This statement is not required and it should only be used when there are both true and false actions (true actions consist of those actions immediately following the condition and continuing until an ELSE or CONTINUE statement is encountered). False actions commence with the first action following the ELSE statement and continue until a CONTINUE statement is encountered.

Example -

...IF THE COUNT IS GREATER THAN 10 COMPUTE COUNT = COUNT *2 ELSE MOVE ZEROS TO COUNT CONTINUE...

In the preceding example, if the condition is true, the COMPUTE statement would be executed. If it is false, the MOVE statement would be executed. Any number of actions could appear where these two actions appear.

9.3.3.5 The CONTINUE Statement

The CONTINUE statement identifies the point, following a condition, that execution of nonconditional statements is resumed. There are no exceptions; each IF statement must have associated with it a corresponding CONTINUE statement. All NFL statements which follow an IF statement are considered to be part of the IF statement. To terminate an IF block, a CONTINUE statement is required.

Example -

...IF FIELD IS NOT EQUAL TO BLANKS MOVE FIELD TO REC CONTINUE...

In the example above, if the condition is true, execution is resumed with the next statement following the CONTINUE after the true actions are performed. If the condition is false, execution is resumed with the next statement following the CONTINUE statement.

Example -

...IF FIELD IS NOT EQUAL TO BLANKS MOVE FIELD TO REC
ELSE MOVE REC TO FIELD CONTINUE...

In the preceding example, if the condition is true, processing is the same as in the previous example. However, if the condition is false, the false actions (those between the ELSE statement and the CONTINUE statement) are executed; execution then is resumed with the next statement following the CONTINUE statement.

9.3.3.6 The Language Identifier Statement

The language identifier statement consists of the characters NFL. It must appear as the first statement following the transaction descriptor deck (or the library action control card if there is no TDD). It identifies the language of all statements between it and the END statement which terminates the last NFL logic statement.

Example -

Library Action Control Card

Transaction Descriptor Deck

NFL

Condition and Action Statements

END.

In the preceding example, the language is identified as NFL. All statements between it (NFL) and the logic statement END card must be written as NFL statements.

POOL statements may be compiled in the same execution as NFL statements. Grouping of NFL statements is recommended.

9.4 NFL Logic Statement Examples

The following examples illustrate the setup of the FM run deck for updating the Logic Statement Library, and the use of some of the NFL language statements. All of the examples pertain to the TEST36Ø file.

All of the sample logic statements, with the exception of the range statement, perform updates with transactions from the report 'RPT36Ø'. The different transaction formats within this report are identified by the letters 'A' through 'G' in column 1 of the transaction.

Comments in the form of NOTE statements are shown in each of the logic statements. The logic statements illustrated in sections 9.4.3 to 9.4.5 and sections 9.4.7 to 9.4.8 are the NFL equivalents of the POOL statements in sections 7.5.3 to 7.5.5 and section 7.5.7 to 7.5.8, respectively. The logic statement in section 9.4.6 is the equivalent of the logic statements in sections 8.1 and 8.2.

9.4.1 FMS Control Card

The following FMS control card would be used to execute the 'LIB' mode of FM to perform updates for the Logic Statement Library for the TEST36Ø file:

```
$FMS/LIB,TEST36Ø
```

9.4.2 Library Action Card to Add a Report

The following card would be used to add the report 'RPT36Ø' to the Logic Statement Library. The transaction ID field, for transactions within this report, is located in column 1.

```
$AR,RPT36Ø,1
```

This card could also have been punched as follows:

```
$AR,RPT36Ø,1-1
```

However, since the transaction ID field is only one byte long, the '-1' is not required.

9.4.3 Logic Statement Setup

The following example illustrates the organization of the library action card, the TDD cards, the language identifier card, and the NFL statement cards for an Exception logic statement. The sample statement performs updates with the 'A' transaction format of the report 'RPT36Ø'.

The first card for the statement is the library action card. This card specifies that the statement is to be permanently added to the library. It also specifies that the statement will perform updates with the 'A' transaction of report 'RPT36Ø', and that the fixed data in that transaction format is 8Ø bytes long. The transaction does not contain any variable data.

The TDD cards follow the library action card. The first field in each of these cards is used to assign mnemonics to the transaction data fields.

The second and third fields specify the high-order position and the low-order position of the transaction fields.

The fourth field is used to specify that a transaction field is a major or user control field. In the example, $RECID is a major transaction control field, and it corresponds to the data record control group, 'UIC'. $SORT is a user transaction control field. It is not used in matching a transaction record to a data record, but is associated with the record control field to control the file processing sequence.

The fifth field in the TDD card indicates the type of data that the transaction fields will contain. The 'A' transaction contains alphabetic (A) data and zoned decimal (D) data only. Insertion of this field is optional, with the default option being 'D'.

FILE MAINTENANCE (FM)

The card following the last TDD card is the language identifier card, and contains the word 'NFL'. The word 'NFL' may appear anywhere between column 1 and column 71, but must be in three consecutive card columns.

The logic statement's function is described in the NOTE statements which follow the language identifier card.

The logic statement first tests the new record switch by using the condition/action statement sequence labeled TEST. A new record will be generated by FM when no data record can be found with a UIC group that matches the contents of the $RECID transaction field in an 'A' transaction. If this has occurred, then the true actions of the condition/action statement will be executed. The true action prints a line which indicates that a new record was generated and control is then passed to the statements following the keyword CONTINUE. If a new record was not generated, control will also be passed to the statements following the keyword CONTINUE because false actions were not specified in this particular condition/action statement sequence.

The MOVE statements move the contents of the transaction fields to data file fields. Transaction data that is moved to coordinate fields will be converted automatically to internal coordinate format. The last MOVE statement will store the date/time of the update into the data field LAUD by referencing the SYSDATE system constant.

The last two condition/action statement sequences are the NFL equivalent of the POOL conditional move instructions. Transaction data $HOME will be moved to data field HOME if it is not blank and $PERS will be moved to PERS if it is not blank.

The END statement is executed next. This statement causes an exit from the logic statement.

```
FILE MAINTENANCE (FM)

$ASP,RPT36Ø,A,8Ø
$RECID,2,7,C1,A
$SORT,8,8,C2,A
$HOME,1Ø,1Ø,,A
$ATTACH,12,12,,A
$FUTURE,13,13,,A
$POINT,15,25,,D
$AREA1,27,37,,A
$AREA2,38,48,,A
$AREA3,49,59,,A
$PERS,73,8Ø,,D
   NFL
NOTE 'THIS LOGIC STATEMENT WILL UPDATE THE LOCATION'
NOTE 'AND DEPLOYMENT AREA OF THE SPECIFIED UNIT.   '
NOTE 'IF A NEW RECORD IS GENERATED, A MESSAGE WILL '
NOTE 'BE PRINTED AND THE TRANSACTION FIELDS WILL BE'
NOTE 'MOVED TO THE DATA FIELDS'
TEST: IF A NEW RECORD
      PRINT 'NEW RECORD GENERATED. ID IS - ',$RECID
      CONTINUE
      MOVE $ATTACH TO ATACH  MOVE $FUTURE TO FUTU
      MOVE $POINT TO POINT MOVE $AREA1 TO DAPT1
      MOVE $AREA2 TO DAPT2 MOVE $AREA3 TO DAPT3
```

```
FILE MAINTENANCE (FM)

     MOVE $AREA4 TO DAPT4 MOVE SYSDATE TO LAUD
     IF $HOME IS NOT EQUAL TO BLANKS
     MOVE $HOME TO HOME  CONTINUE
     IF $PERS IS NOT EQUAL TO BLANKS
     MOVE $PERS TO PERS CONTINUE
END
```

9.4.4 Use of Data Conversion

The following logic statement is used with 'B' transactions. These transactions contain only a major control field in positions 2 through 7.

This logic statement verifies that the CNTRY and ACTIV fields, in selected records, contain valid data.

If a new record is generated by a 'B' transaction, the record will be deleted and an exit from the logic statement will be taken when the RETURN statement is executed.

If a new record is not generated, control is passed to the statements following the keyword CONTINUE. Three work areas are then defined to hold the results of the conversion routines or the asterisks error flag. Data field CNTRY will be moved to the work area CNTBUC if the conversion by table CTRYS is successful. If the conversion is unsuccessful, an exit to the statement labeled ERR1 will occur and asterisks will be moved to the work area CNTBUC. The statement labeled ACT performs a similar function with the data field ACTIV. Before the logic statement exits, the results of the two conversions will be printed. The system word BLANK in the PRINT statement will insert a single blank character in the printed line.

FILE MAINTENANCE (FM)

```
$ASP,RPT360,B,80
$RECID,2,7,C1,A
NFL
NOTE ' THIS LOGIC STATEMENT WILL EXTRACT THE COUNTRY '
NOTE ' CODE AND THE ACTIVITY CODE FOR SPECIFIED UNITS'
  IF A NEW RECORD DELETE THE RECORD RETURN  CONTINUE
    DEFINE CNTBUC AS #15  DEFINE ACTBUC AS #15
    DEFINE ASTRK AS '***************'
    MOVE CNTRY TO CNTBUC USING TABLE CTRYS ,EXIT TO ERR1
    GO TO ACT
ERR1: MOVE ASTRK TO CNTBUC
ACT:  MOVE ACTIV TO ACTBUC USING TABLE ACTVS, EXIT TO ERR2
      GO TO PRT
ERR2: MOVE ASTRK TO ACTBUC
PRT:  PRINT UIC,BLANK,CNTBUC,BLANK,ACTBUC
         END
```

9.4.5 Periodic Set Processing

The following examples illustrate two methods for updating periodic subsets. The first example uses an Exception update statement to perform the updating of the record. The library action card will add statement 'D' of report type 'RPT36Ø' permanently to the logic statement library. The TDD cards assign mnemonics to the transaction fields, with transaction field $RECID containing the major control field of the record to be processed.

In the first example, the POSITION AFTER LAST statement will ensure that the data file is at the beginning of the subsets. The statement labeled POS will cause the subset of periodic set one to be searched for the data field MECLQ being equal to the contents of the transaction field $MEQPT. If found, the following IF statement will modify data field MEPSD and exit from the logic statement because RETURN statements are in both the true and false actions part of the condition/action statement sequence. If the subset is not found, then an exit will be taken to the statement labeled NEW, where a new subset will be created, transaction data will be set into the specified fields and a message will be printed to indicate this action.

In the second example, the library action card is the same. The TDD card for the transaction field '$MEQPT' is different for the direct subset update. The control parameter on the TDD card indicates that the field is to be used as a subset control field. It also carries a corresponding data record subset control field parameter. The data field name 'MECLQ' is defined as a subset control group in the FFT. If no subset exists with a total record control group (major control field, set number, and subset control group) equal to the update record control group ($RECID, set number, $MEQPT) a new subset is generated by FM and the total record control group is set in the new subset and the new record indicator is set on. If the subset exists, the subset is made active.

```
$ASP,RPT36Ø,D,8Ø

$RECID,2,7,C1,A
```

```
FILE MAINTENANCE (FM)

$MEQPT,10,22,,A
$NOEQPT,25,27,,D
$ADDCODE,29,29,,A
    NFL
NOTE 'THIS LOGIC STATEMENT WILL SEARCH FOR THE SUBSET'
NOTE 'CONTAINING THE EQUIPMENT TYPE AND WILL ADD OR  '
NOTE 'SUBTRACT THE NUMBER OF ITEMS AS SPECIFIED BY THE'
NOTE 'ADD CODE. IF THE SUBSET DOES NOT EXIST, A NEW'
NOTE 'SUBSET WILL BE BUILT AND THE FIELDS UPDATED'
    POSITION AFTER LAST SUBSET IN MECLQ
 NOTE 'ABOVE STATEMENT FORCES SET INACTIVE'
POS:   POSITION TO $MEQPT IN MECLQ, EXIT TO NEW
       IF $ADDCODE EQUALS 'A'
          COMPUTE MEPSD = MEPSD + $NOEQPT , RETURN
       ELSE
          COMPUTE MEPSD = MEPSD - $NOEQPT,RETURN
       CONTINUE
NEW: BUILD SUBSET FOR MECLQ  MOVE $MEQPT TO MECLQ
     MOVE $NOEQPT TO MEPSD
     PRINT 'NEW SUBSET CREATED'  PRINT $MEQPT
 END
```

```
FILE MAINTENANCE (FM)

$ASP,RPT36Ø,D,8Ø
$RECID,2,7,C1,A
$MEQPT,1Ø,22,,A,S,MECLQ
$NOEQPT,25,27,,D
$ADDCODE,29,29,,A
  NFL
NOTE ' THIS STATEMENT WILL PERFORM THE SAME FUNCTION, '
NOTE ' USING THE DIRECT SUBSET UPDATE CAPABILITY.'
NOTE ' IF THE SUBSET DOES NOT EXIST A NEW SUBSET WILL '
NOTE ' BE GENERATED AND THE SUBSET CONTROL FIELD '
NOTE ' WILL BE AUTOMATICALLY SET BY FM. '
    IF A NEW RECORD MOVE $NOEQPT TO MEPSD
      PRINT 'NEW SUBSET CREATED'  PRINT $MEQPT
      RETURN  CONTINUE
    IF $ADDCODE IS EQUAL TO 'A'
       COMPUTE MEPSD = $NOEQPT + MEPSD, RETURN
    ELSE
       COMPUTE MEPSD = $NOEQPT - MEPSD,RETURN  CONTINUE
    END
```

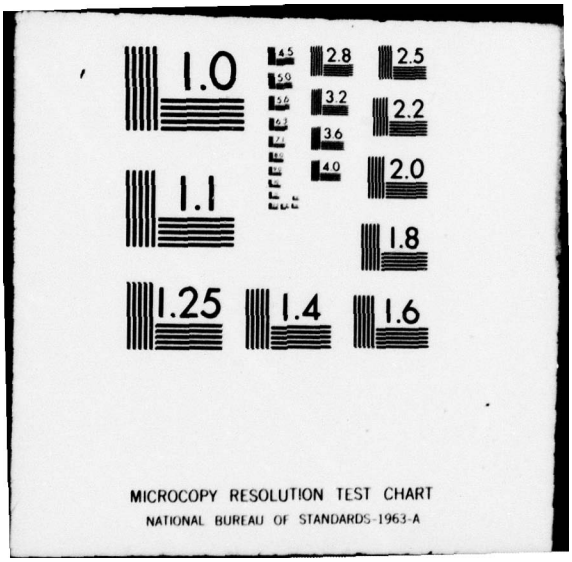
9.4.6 Test for Numeric Data

The following logic statement illustrates a simple method of determining if a transaction field contains zoned decimal data. The PICTURE test is used for this function. The PICTURE mask will contain the letter N which signifies a numeric character test. If all characters of the field are to be tested, then the mask will contain a number of Ns equal to the length of the field. This statement consists of condition/action statement sequences containing both true and false actions. If the transaction data contains all numeric characters, then it is moved to the data field. If not, an error message is printed.

AD-A063 432 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 9/2
NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (N--ETC(U)
SEP 78 C K HILL
UNCLASSIFIED CCTC-CSM-UM-15-78-VOL-3 NL

3 OF 3
AD
A063 432

END
DATE
FILMED
3-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
FILE MAINTENANCE (FM)

$ASP,RPT36Ø,E,8Ø
$RECID,2,7,C1,A
$PERSONL,1Ø,15,,D
$READAVG,2Ø,22,,D
$RITNM,25,27,,D
   NFL
NOTE ' THIS LOGIC STATEMENT WILL UPDATE NUMERIC FIELDS IN'
NOTE ' THE FIXED SET. THE INPUT TRANSACTION FIELDS ARE IN'
NOTE ' ZONED DECIMAL FORM. CHECKS WILL BE MADE TO DETERMINE'
NOTE ' IF THE FIELDS CONTAIN NUMERIC CHARACTERS. IF AN ERROR'
NOTE ' IS DETECTED, AN ERROR MESSAGE AND THE FIELD IN'
NOTE ' ERROR WILL BE PRINTED'
   IF $PERSONL PICTURE IS 'NNNNNN' MOVE $PERSONL TO PERS ELSE
   PRINT 'THE PERSONL FIELD CONTAINS NON-NUMERIC CHARACTERS'
   $PERSONL CONTINUE   IF $READAVG PICTURE IS 'NNN' MOVE
   $READAVG TO READAVG ELSE
   PRINT 'THE READAVG FIELD CONTAINS NON-NUMERIC CHARACTERS'
   $READAVG CONTINUE IF $RITNM PICTURE IS 'NNN' MOVE $RITNM
   TO RITNM  ELSE PRINT
   'THE RITNM FIELD CONTAINS NON-NUMERIC CHARACTERS' $RITNM
   CONTINUE END
```

9.4.7 Production of Summary Information

The following logic statement is a Range statement that functions without transaction data. The library action code for this statement contains only the function code '$AST', and there are no TDD cards for this statement. This type of statement must be compiled on-line each time it is used.

This statement uses the job complete test to determine if the processing is complete. If processing is not complete, three COMPUTE statements update the desired statistics. When processing is complete, these statistics will be printed.

FILE MAINTENANCE (FM)

```
$AST
  NFL
NOTE ' RANGE STATEMENT TO CALCULATE THE TOTAL NUMBER OF'
NOTE ' UNITS IN THE DATA FILE, TOTAL PERSONNEL STRENGTH'
NOTE ' AVERAGE OF TOTAL READINESS AVERAGE OF ALL UNITS'
NOTE ' THIS INFORMATION WILL BE PRINTED ON THE'
NOTE ' AUXILIARY OUTPUT FILE'
    DEFINE COUNT AS B1,DEFINE TOTPER AS B2,DEFINE AVG AS B3
    IF THE JOB IS NOT COMPLETE  COMPUTE COUNT = COUNT + 1
       COMPUTE TOTPER = TOTPER + PERS,COMPUTE AVG = AVG +
       READAVG RETURN CONTINUE
    PRINT BLANKS PRINT BLANKS   NOTE 'SPACE TWO LINES'
    PRINT 'TOTAL NUMBER OF UNITS-',COUNT  PRINT BLANKS
    PRINT 'TOTAL PERSONNEL STRENGTH OF ALL UNITS-' TOTPER
    PRINT BLANKS  DEFINE TOTAVG AS #12
    COMPUTE TOTAVG = AVG / COUNT
    PRINT 'AVERAGE OF TOTAL READINESS AVERAGE-' ,TOTAVG
    PRINT BLANKS  END
```

9.4.8 Variable Field and Variable Set Processing

The following logic statements illustrate the use of the MOVE statement when the variable field or variable set are referenced.

The first statement moves information from a variable transaction field, $VAR, to the variable field COMMENT. Existing information in COMMENT will be destroyed. When the data transfer takes place, the data is truncated so that any trailing blanks in the variable transaction field are not moved.

The second statement appends information to the variable set REFER. Since the information to be transferred is in a fixed length transaction field, no truncation takes place.

```
FILE MAINTENANCE (FM)

$ASP,RPT360,F,10,11
$RECID,2,7,C1,A
$AVR,11
          NFL
NOTE ' THIS LOGIC STATEMENT REPLACES THE INFORMATION'
NOTE ' IN THE VARIABLE FIELD COMMENT WITH THE '
NOTE ' INFORMATION IN THE VARIABLE LENGTH TRANSACTION'
NOTE ' FIELD $VAR. '
      MOVE $VAR TO COMMENT
      END

$ASP,RPT36Ø,G,8Ø
$RECID,2,7,C1,A
$VAR,31,8Ø,,A
      NFL
NOTE ' THIS LOGIC STATEMENT APPENDS THE INFORMATION IN'
NOTE ' THE TRANSACTION FIELD $VAR TO THE INFORMATION'
NOTE ' IN THE VARIABLE SET REFER.'
      ATTACH $VAR TO REFER
      END
```

9.5 Summary of NFL Condition and Action Statement Syntax

The following shows the syntax format for NFL conditional and action statements.

The following legends are used:

TF	-Transaction Field Name
DF	-Data File Name
IA	-Indirect Address Name
WA	-Work Area Name
LV	-Literal Value
PF	-Partial Field Value
SN	-Set Number
FC	-Figuration Constant (SYSDATE, ZERO, ZEROS, ZEROES, BLANK, BLANKS)
ST	-Subroutine or Table Name
SL	-Symbolic Statement Name
PM	-Picture Mask designated as an alpha LV
BM	-Bit Mark designated as an unsigned numeric LV consisting of ones and zeros
[ ]	-Optional Words
	-Choose One Word

Note: Partial field notation for figurative constants is valid only for SYSDATE.

IF Statement (Non-between Relational)

```
                                  EQ              TF[PF]      TF[PF]
                                  NE              DF[PF]      DF[PF]
                                : EQUAL  [TO]     WA[PF]      WA[PF]
                                : EQUALS          FC[PF]      FC[PF]
                                :                 IA          IA
                                :                 LV          LV
                                :
                                :                 MULTIPLE OPERANDS
                                                  PERMITTED
                                :
                                :
                                :
                                :
    TF[PF]                      :
    DF[PF]                      :
IF  WA[PF]  [IS][NOT]           :
    IA                          :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                :
                                : LT                TF[PF]
                                : GT                DF[PF]
                                : LE                WA[PF]
                                : GE       [THAN]   FC[PF]
                                  LESS              IA
                                  GREATER           LV

                                       SINGLE OPERAND ONLY
```

FILE MAINTENANCE (FM)

IF Statement (Between Relation)

```
                                                TF[PF]     TF[PF]
        TF[PF]                          BT      DF[PF]     DF[PF]
        DF[PF]                                  WA[PF] /   WA[PF]
IF      WA[PF]  [IS] [NOT]  BETWEEN             FC[PF]     FC[PF]
        IA                                      IA         IA
                                                LV         LV
```

MULTIPLE OPERANDS PERMITTED

BETWEEN OPERAND FORMAT:

NO PARTIAL FIELD -	MEQPT/SERV or MEQPT / SERV
WITH PARTIAL FIELD -	MEQPT 1/2 / SER 3/4
	MEQPT / SERV 3/4
	MEQPT 1/2 / SERV

IF Statement (Table Relation)

```
        TF[PF]                          TAB
IF      DF[PF]   [IS] [NOT] [IN]              ST
        WA[PF]                          TABLE
        IA
```

IF Statement (Picture Relation)

```
        TF[PF]      PIC
IF      DF[PF]                      [IS] [NOT] PM
        WA[PF]      PICTURE
        IA
```

IF Statement (Bit Test)

```
     TF[ PF]
     DF[ PF]                                ON
IF   WA[ PF]  BIT BM [ IS] [NOT]
     IA                                     OFF
```

IF Statement (Switch Test)

```
                              ON
IF   WA   [IS] [NOT]
                              OFF
```

IF Statement (Status Test)

```
IF [NOT] NEW RECORD

                              COMPLETE
IF [THE] JOB [IS] [NOT]
                              COMPLETED

                              ON
IF OVERFLOW [IS] [NOT]
                              OFF
```

FILE MAINTENANCE (FM)

ACTION Statements

```
GO          [TO]    SL

                    ON
TURN          WA
                    OFF

              DF                TF[PF]      +     TF[PF]
COMPUTE       WA      =         DF[PF]      -     DF[PF]
              IA                WA[PF]      *     WA[PF]
                                FC[PF]      /     FC[PF]
                                IA                IA
                                LV                LV

WRITE                           TF[PF]      TF[PF]
PRINT           [ON n]          DF[PF]      DF[PF]
PUNCH                           WA[PF]      WA[PF]
DISPLAY                         FC[PF]      FC[PF]
                                LV          LV
```

n = unit number

for PRINT, PUNCH, DISPLAY n may be 1 or 2

for WRITE n may be 1 through 5

If [ON n] not specified, = unit 1 is assumed.

```
                DF
BUILD SUBSET    IA
                SN
```

```
                TF[PF]
                DF[PF]              DF
ATTACH          WA[PF]    [TO]
                FC[PF]
                IA
```

FILE MAINTENANCE (FM)

```
            LV


DELETE RECORD

            FIELD          DF
DELETE      SUBSET         IA
            SET            SN


          TF[PF]                                    SUB
          DF[PF]                    DF[PF]          TAB
MOVE      WA[PF]  [TO]              WA[PF]  [USING]   SUBROUTINE   ST EXIT SL
          FC[PF]                    IA              TABLE
          IA
          LV


                    DF
LOCATE SET          IA      EXIT SL
                    SN


                  DF
STEP SET          IA      EXIT SL
                  SN
```

FILE MAINTENANCE (FM)

```
                        TF[PF]
                        DF[PF]                  DF[PF]
POSITION [TO]           WA[PF]  [IN]                        EXIT SL
                        IA                      IA
                        LV

POSITION [TO] FIRST [SUBSET] [IN]  DF
                                   IA           EXIT SL
                                   SN

POSITION [TO] NEXT [SUBSET] [IN]   DF
                                   IA       EXIT SL
                                   SN

POSITION [AFTER] LAST [SUBSET] [IN]  DF
                                     IA    EXIT SL
                                     SN

              AS
DEFINE WA             LV
              TO

                        Wn/m
              AS                                    FC
DEFINE  WA              #n        VALUE [IS)
              TO                                    LV
                        Bn
```

Appendix

Utilizing a NIPS File as FM Transaction Input

This appendix specifies the preparation requirements for using a NIPS 36Ø FFS data base as a transaction input file.

The high-order location, length and mode of each file field may be obtained from the File Format Record List portion of the FFT listing. These high-order locations are relative Ø and must be adjusted to relative 1 (add 1 to each H.O. location) for FM TDD cards. If in doubt, run an FR and list the logic statements produced. The TDD cards of these statements will also describe each field in the file being revised.

If the file has periodic sets, a logic statement can be written for each set. The LS name would be formed from two bytes. The first byte would be an 'R' which is in position 6 (relative to 1) of every data record. The second would be one hex byte which specifies the set I.D. The location of this byte can be found under the label 'H.O. SET ID' in the Control Record Contents portion of the FFT list. Again, this is relative to Ø; so add one for the TDD card. This byte is a binary Ø for the fixed set, binary 1 for the first periodic set, binary 2 for the second, etc.

The TRANS DD statement in the FM procedure should be overridden to describe the NIPS file. The DCB for a NIPS ISAM file must include the parameter DSORG=IS.

The transaction source field on the FM control card should be specified as SAM for a sequential NIPS data base or ISAM for an index sequential NIPS data base.

Using file TEST36Ø as an example, the record control is six bytes long which places the SET ID at location 13 of each file record.

a. The Add Report card would appear as follows:

$AR,REPT,6,13

b. The Add Statement card for the fixed set logic statement would appear as follows:

$ASP,REPT,R*,32Ø

where * is a binary Ø (12-Ø-8-1 punch) and 32Ø is the set length.

DISTRIBUTION

CCTC CODES	COPIES
C124 (Reference and Record)	3
C124 (Record Copy) Stock	6
C240	20
C315	1
C341 (Maintenance Contractor)	10
C341 (Stock)	70

EXTERNAL

Director of Administrative Services, Office of the Joint Chiefs of Staff Attn: Chief, Personnel Division, Room 1A724, The Pentagon Washington, D.C. 20301	1
Director for Personnel, J-1, Office of the Joint Chiefs of Staff, Attn: Chief, Data Service Office, Room 1B738C, The Pentagon, Washington, D.C. 20301	1
Director for Operations, J-3, Office of the Joint Chiefs of Staff, Attn: P & AD, Room 2B870, The Pentagon, Washington, D.C. 20301	1
Director for Operations, J-3, Office of the Joint Chiefs of Staff, Attn: Deputy Director for Operations (Reconnaisance and Electronic Warfare) Room 2D921, The Pentagon, Washington, D.C. 20301	1
Director for Logistics, J-4, Office of the Joint Chiefs of Staff, Room 2E828, The Pentagon, Washington, D.C. 20301	1
Chief, Studies Analysis and Gaming Agency, Attn: Chief, Force Analysis Branch, Room 1D928A, The Pentagon, Washington, D.C. 20301	1
Automatic Data Processing, Liaison Office National Military Command Center, Room 2D901A, The Pentagon, Washington, D.C. 20301	1

EXTERNAL	COPIES
Automatic Data Processing Division Supreme Headquarters Allied Powers, Europe Attn: SA & P Branch, APO New York 09055------	1
Director, Defense Communications Agency, Office Of MEECN System Engineering, Attn: Code 960T, Washington, D.C. 20301----------	1
Director, Defense Communications Engineering Center, Hybrid Simulation Facility, 1860 Wiehl Avenue, Reston, VA 22070----------	1
Director, Defense Intelligence Agency Attn: DS - 5C2 Washington, D.C. 20301----------	5
Commander-in-Chief, Pacific, Attn: J6331, FPO San Francisco, 96610----------	1
Commander-in-Chief, US Army Europe and Seventh Army ATTN: OPS APO New York 09403---	1
Commanding General, US Army Forces Command, Attn: Data Support Division, Building 206, Fort McPherson, GA 30303----------	1
Commander, Fleet Intelligence Center, Europe, Box 18, Naval Air Station, Jacksonville, Florida 32212----------	1
Commanding Officer, Naval Air Engineering Center, Ground Support Equipment Department, SE 314, Building 76-1, Philadelphia, PA 19112	1
Commanding Officer, Naval Security Group Command, 3801 Nebraska Avenue, N.W. Attn: GP22, Washington, D.C. 20390----------	1
Commanding Officer, Navy Ships Parts Control Center, Attn: Code 712, Mechanicsburg, PA 17055	1
Headquarters, US Marine Corps, Attn: System Design and Programming Section (MC-JSMD-7) Washington, D.C. 20380----------	1

EXTERNAL	COPIES
Commanding Officer, US Army Forces Command Intelligence Center, Attn: AFIC-PD, Fort Bragg, NC 28307----------	1
Commander, US Army Foreign Science and Technology Center, Attn: AMXSJ-CS, 220 Seventh Street NE, Charlottsville, VA 22212--	1
Commanding Officer, US Army Security Agency, Command Data Systems Activity (CDSA) Arlington Hall Station, Arlington, VA 22212----------	1
Commanding Officer, US Army Security Agency Field Station - Augsburg, Attn: IAEADP, APO New York 09458----------	1
Commander, Fleet Intelligence Center, Atlantic, Attn: DPS, Norfolk, VA 23511----------	1
Commander, Fleet Intelligence Center, Pacific, Box 500, Pearl Harbor, HI 96860----------	1
Air Force Operations Center, Attn: Systems Division (XOOCSC) Washington, D.C. 20301-----	1
Commander, Armed Forces Air Intelligence Training Center, TTMNIM (360 FFS), Lowry AFB, Co 80230----------	1
Commander, Air Force Data Services Center, Attn: Director of System Support, Washington, D.C. 20330----------	1
Commander-in-Chief, US Air Forces in Europe, Attn: ACDI APO New York 09332----------	1
Commander, USAF Tactical Air Command, Langley AFB, VA 23665----------	1
Commander, Space and Missile Test Center, Attn: (ROCA) Building 7000, Vandenberg, AFB, CA 93437----------	1

EXTERNAL	COPIES
Naval Air Systems Command, Naval Air Station, Code 13999, Jacksonville, Florida 32212------	1
Commanding General, US Army Computer Systems Command, Attn: Support Operations Directorate, Fort Belvoir, VA----------	1
Defense Documentation Center, Cameron Station, Alexandria, VA 22314----------	12
TOTAL	159

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: CSM UM 15-78 Volume III 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): NMCS Information Processing System 360 Formatted File System (NIPS 360 FFS) - Users Manual Vol III - File Maintenance (FM)	5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s)	8. CONTRACT OR GRANT NUMBER(s): DCA 100-77-C-0065
9. PERFORMING ORGANIZATION NAME AND ADDRESS: International Business Machines, Corp. Rosslyn, Virginia	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: National Military Command System Support Center The Pentagon, Washington, D.C. 20301	12. REPORT DATE: 1 September 1978 13. NUMBER OF PAGES: 209
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Copies of this document may be obtained from the Defense Documentation Center, Cameron Station, Alexandria, Virginia 22314.
This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This volume defines the File Maintenance (FM) component of NIPS 360 FFS. It describes the functioning of the component, its capabilities, limitations, expected output results, and specifications for preparing run decks and control cards which will serve as reference for the knowledgble user.

This document supersedes CSM UM 15-74, Volume III.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)